Universidad Nacional de La Plata.
Facultad de Informática.

Tesis de Grado

Estudio del problema de School Timetabling utilizando técnicas de Inteligencia Artificial

Pablo Ezequiel Sánchez

acpablosanchez@yahoo.com.ar

Directora
Dra. Clara Smith.

Octubre de 2008

Contenido

Capítulo 1 - Presentación

1.1 Introducción

Los problemas de *school timetabling* consisten en asignar grupos de estudiantes a clases de distintas materias escolares, donde cada clase ocurre en un cierto período de tiempo, y es enseñada por un profesor dado. El resultado es *Timetable* armado u "horario de clases".

Usar el modo manual para generar horarios de clase es una tarea muy dura, particularmente porque hay demasiados conflictos que resolver (por ejemplo, superposiciones horarias). Trabajar manualmente involucra numerosas idas y vueltas y cambios antes de lograr un horario satisfactorio. Normalmente el proceso toma varios días o semanas, y la calidad del horario final que se obtiene como resultado está dada principalmente por la destreza de quien lo realiza.

Generar horarios automáticamente aparece como una alternativa atractiva a la forma manual tradicional. Sin embargo esta aproximación no está libre de problemas, ya que construir programas computacionales que logren horarios que cumplan con unas pocas restricciones nos eleva al tratamiento de problemas que pertenecen a la clase NP [De Werra, 1985]. La clase de problemas NP contiene problemas para los cuales se desconocen algoritmos en tiempo polinomial, pero para los cuales sí hay soluciones en tiempo polinomial si se nos permite "predecir" una solución y si no somos "penalizados" por predecir soluciones que luego son fallidas; en otras palabras, si sólo son tenidas en cuenta las soluciones que llevan a una respuesta satisfactoria [Weiss, 1997].

De hecho, la mayoría de los problemas de *Timetabling* son NP-Completos, por lo que muchos investigadores están interesados en encontrar algoritmos eficientes para resolverlos.

El *Timetabling Problem* es NP-Completo [Even et al, 1976].

1.2 Caso de Estudio

La motivación de este trabajo surge de la generación del horario semanal de clases en el colegio secundario E.E.M. Nº 2 de la localidad de Henderson, Pcia. de Bs. As. El mismo se realiza actualmente en forma manual por un encargado. Esta tarea suele llevarle al empleado todo un verano, pues debe tener en consideración múltiples condiciones de bandas horarias y superposiciones de cursos.

El mismo encargado nos contactó personalmente para ver la posibilidad de realizar un programa de computadora que pudiera asistirlo en tal tarea, y en lo posible, resolver completamente la asignación horaria de clases del colegio.

La tarea comienza con una lista de cursos que se dictan en el colegio, y con la lista de horarios preferidos de los profesores que dictan materias para esos cursos. El encargado toma esas listas y asigna las horas disponibles de la grilla semanal a cada una de las materias, de forma tal que un mismo profesor no dicte clases en mas de un curso al mismo tiempo y que dicte preferiblemente en los horarios que eligió previamente. Tampoco puede ocurrir que un curso reciba el dictado de dos materias al mismo tiempo, es decir el mismo día a la misma hora. A esto se le pueden sumar eventualmente tantas restricciones como lo dispongan las autoridades del colegio.

Además de estas restricciones impuestas en el caso de estudio, en nuestro análisis contemplamos también las condiciones que debe tener en cuenta cualquiera de las técnicas que utilizaremos:

Restricciones Duras: (no pueden dejarse de lado)

- No asignar dos lecciones al mismo *timeslot*
- Un profesor no puede dictar en dos cursos al mismo tiempo.

Restricciones opcionales / Livianas:

- Preferencias de horarios por parte de los profesores.
- Máximo en la cantidad de horas diarias que se puede dictar una materia a un curso.

- Posibilidad o no de dejar *timeslots* sin asignación de materia, es decir, tener o no horas libres.

Por todas estas restricciones es que el encargado lleva una “calificación” de cada profesor para saber con cuáles comenzar la asignación. Esta “calificación” se basa en factores como antigüedad, cantidad de materias y horas, responsabilidad, si dicta o no en varios colegios, flexibilidad en cuanto a posiblidad de cambio de horas asignadas y libres, y otras “ideas” sobre el profesor para asegurar la consistencia del Timetable.

1.3 Objetivos

Los objetivos de esta Tesis de Grado son:

Objetivo Principal:

Analizar la utilidad de mecanismos automáticos existentes para la generación de horarios adecuados a la escuela que nos convoca, sugiriendo alternativas y eventuales perfeccionamientos a los mismos.

Objetivos Específicos:

1)Modelar como caso de estudio la escuela citada de modo de que el resultado de esa modelización pueda ser utilizado como entrada para el cálculo y estudio de algoritmos de *timetabling*. Para ello investigamos lenguajes y estándares en la codificación del conocimiento asociado al *Timetabling Problem*. Entre otras tareas, por ejemplo, logramos una forma normal conjuntiva para el caso, de modo de poder testear SAT sobre ella.

2)Estudiar cómo se comportan diferentes algoritmos y métodos cuando son aplicados sobre el caso de estudio. Esto lo hacemos tanto desde el punto de vista teórico como práctico.

3)Analizar los resultados en función de métricas y estándares de la comunidad científica dedicada a la investigación del *timetabling problem*.

4) Obtener Conclusiones que permitan decidir el uso de una u otra técnica de Inteligencia Artificial para la resolución del *Timetabling* Problem, en especial para la escuela en estudio.

1.4 Implementación y experiencias

El relevamiento de la bibliografía y material específico sobre la asignación de horarios nos permitió comprobar que casi todos los papers en la literatura describen posibles implementaciones en software para la solución del *Timetabling Problem*, siempre enfocadas a un caso particular. Los autores coinciden en que no existe un modelo general que pueda ser aplicado a todas las realidades.[Abramson,1991] [Colorni,1992] [Costa,1994] [Kenneth-chin Fat, 2004] [Redl, 2004] [Lalescu, 2003] y [Schaerf, 1995].

Estos mismos papers nos sirvieron de guía para realizar una implementación genérica que utilizara diferentes técnicas para nuestra Escuela Secundaria de test.

El éxito de una implementación de este tipo se mide de dos diferentes modos, dependiendo de si la aplicación resuelva un *search problem* o un *optimization problem*. En el primer caso, la medida del éxito es el número de lecciones asignadas con respecto al total para asignar. Los reportes de los resultados varían entre el 95% y el 100% (es decir, que se pudo asignar todo). Tales valores están obviamente influenciados por cómo está restringida la instancia particular, por eso no hay una medida exacta de la calidad de un programa de este tipo.

En el segundo caso, la medida del éxito está dada por el valor de la función objetivo para la solución óptima. Tal valor tiene un significado práctico directo; por ejemplo: el número de profesores satisfechos con el horario asignado.

En ambos casos, los resultados se comparan con los hechos “a mano”. Se puede decir que los resultados producidos por una computadora son siempre (o mayoritariamente) superiores a los hechos “a mano”.

La complejidad computacional de los algoritmos aquí estudiados se determina por el tiempo de computación. En la mayoría de los casos, tal tiempo está en el orden de los segundos. El uso de PC es aconsejable para el school timetabling porque es un tipo de computadora que puede efectivamente tener una oficina administrativa de un Colegio Secundario.

1.5 Organización de la Tesis

En el segundo capítulo mostramos la complejidad del problema de *timetabling* y algunas variantes propuestas por la literatura y la comunidad. Las variantes se basan en el tipo de institución involucrada (universidad o colegio secundario) y el tipo de restricciones a tener en cuenta. También analizamos el Estado del Arte con respecto a las técnicas o aproximaciones a la solución del mismo.

En el tercer capítulo presentamos la información que maneja el encargado de la Escuela Secundaria para armar el *Timetable,* y la forma manual que utiliza para alcanzar ese objetivo.

En los Capítulos 4, 5 y 6 estudiamos y ejemplificamos las técnicas:

- Técnicas de reducción a la coloración del grafo. [Redl,2004] propone una reducción del timetabling problem al problema de coloración del grafo.
- Técnicas de Algoritmos Genéticos aplicados al problema de school timetabling [Lalescu,2003].
- Técnicas SAT (Satisfactibilidad). Tomaremos como base el método de [Kenneth-Chin Fat,2004].

Estas técnicas proveerán base suficiente para una comprensión inicial de mecanismos de solución al problema de *timetabling*.

El Capítulo 7 que es el central de la Tesis. Nos abocamos a:

aplicar en forma práctica los diferentes algoritmos y métodos sobre el Caso de Estudio. Analizamos luego los resultados en función de métricas y estándares de la comunidad científica dedicada a la investigación del *timetabling problem.*

Finalmente damos nuestras Conclusiones sobre la base de las comparativas de las técnicas aplicadas y sus resultados.

Capítulo 2 – Estado del Arte en Timetabling

2.1 Introducción

Para poder lograr una comparación efectiva de las técnicas que entreguen solución al problema de *Timetabling*, es necesario conocer las investigaciones que se han realizado sobre el tema, viendo entre otras cosas las variables que afectan directamente en éste.

Por lo tanto, describiremos en qué consiste el problema de *Timetabling*, cómo se clasifican estos tipos de problemas, cuáles son las variables que influyen y las técnicas que se pueden utilizar.

2.2 Complejidad

La clase NP

Un poco por debajo de los horrores de los problemas indecidibles se encuentra la clase NP. NP significa tiempo polinómico no determinista (*nondeterministic polynominal-time*) [Weiss, 1997].

Una máquina determinista, en cada instante está ejecutando una instrucción. Usualmente, después pasa a alguna otra instrucción, que es única. Una máquina no determinista tiene en un punto dado de la ejecución, una variedad de pasos siguientes alternativos. Es libre de escoger el que quiera, y si uno de esos pasos lleva a una solución, siempre elegirá el correcto. Así, con esta ficción de computación, no somos penalizados por elegir caminos incorrectos.

Una forma sencilla de comprobar si un problema está en la clase NP es expresar el problema como una pregunta sí/no. El problema está en NP si, en tiempo polinómico, se puede demostrar que cualquier ocurrencia “si” es correcta. No tenemos que preocuparnos de las ocurrencias “no”, ya que el programa siempre hace la elección correcta.

Problemas NP completos

Entre todos los problemas de los que se sabe que están en NP, hay un subconjunto, llamado problemas NP completos [De Werra, 1985]. Un problema NP completo tiene la propiedad de que cualquier problema en NP puede ser reducido polinómicamente a él.

Un problema P1 puede ser reducido a P2 como sigue: se hace una correspondencia tal que cualquier ocurrencia de P1 pueda transformarse en una ocurrencia de P2. Se resuelve P2 y después se hace una correspondencia de la respuesta del nuevo con el original.

Un problema que se demostró que era NP completo fue el de *satisfacibilidad SAT* para Lógica de Enunciados. El problema de satisfacibilidad toma como entrada una expresión booleana y pregunta si la expresión tiene una asignación a las variables que dan un valor de 1 (True).

En 1971 Cook demostró que satisfacibilidad (SAT) era NP completo demostrando directamente que todos los problemas que están en NP se pueden transformar a SAT. Para ello se valió del hecho conocido acerca de todo problema NP: todo problema en NP puede resolverse en tiempo polinómico por medio de un computador no determinista. El modelo formal de un computador se denomina **máquina de Turing**. Cook mostró cómo se pueden simular las acciones de esta máquina con una fórmula booleana, extremadamente larga y compleja, pero polinómica. Esta fórmula booleana sería verdadera si y sólo si el programa que ejecutara la máquina de Turing produjera una respuesta “sí” a su entrada.

2.3 Problema de Timetabling

Los problemas de Timetabling pueden verse como problemas de “alocación de recursos”, se asocian a la labor de organizar una secuencia de eventos (generalmente materias o exámenes), en un período de tiempo determinado, satisfaciendo un conjunto de restricciones. Las restricciones comprenden hechos tales como evitar los choques de horarios, capacidad de salas, carga de trabajo y disposición para estudiantes y profesores, entre otros [Schaerf, 1995].

Aún cuando la definición y terminología varía de una institución a otra, existen conceptos que tienen en común:

1. Clases: corresponde a las reuniones que se realizan, donde se imparte el contenido de la materia. Dependiendo de las características de la materia, puede ser dictada una o más veces a la semana, dando la posibilidad de tener diferentes profesores durante la semana.
2. Curso: se les denomina al grupo de estudiantes que toman las mismas materias y usualmente permanecen juntos durante la semana.
3. Materia: cada una de las asignaturas que cada estudiante toma en el año escolar.
4. Programa: el conjunto de materias organizadas para ser cursadas por los estudiantes.
5. Bloque (*Timeslot*): es el período de tiempo, compuesto por día y hora de la semana, que va a ser asignado a una clase.
6. Horario (*Timetable*): es el conjunto de todos los bloques que componen una semana asignados a diferentes clases para cada curso.

Tipos de Problemas de Timetabling

Se han propuesto un gran número de variantes del problema de timetabling, que difieren unas de otras en el tipo de institución involucrada (Universidad o Colegio Secundario) y el tipo de restricciones (*constraints*) que el timetable debe respetar.

El material bibliográfico consultado nombra como referente del estudio del *Timetabling* al trabajo de Schaerf [Schaerf, 1995] quien clasifica los problemas de timetabling en tres grupos:

School Timetabling: se llama a la asignación de horarios semanales para los cursos de un colegio secundario, evitando que los profesores dicten en dos cursos al mismo tiempo, y viceversa;

University Timetabling: la asignación de horarios semanales de un conjunto de clases de diferentes materias de la Universidad, minimizando la superposición de clases entre años que comparten estudiantes.

Examination Timetabling: la organización de exámenes de materias de la Universidad, evitando solapar exámenes del mismo año.

Viabilidad y Optimización

En algunos casos, el problema de timetabling consiste en encontrar cualquier horario que satisfaga todas las restricciones. En estos casos, el problema se formula como un *Search Problem* (problema de búsqueda). [Schaerf, 1995].

En otros casos, el problema se formula como un *Optimization Problem* (problema de optimización). Esto es, lo que se requiere es un horario que satisfaga todas las *hard constraints* (restricciones duras) y minimice (o maximice) una función objetivo que mide las s*oft constraints* (restricciones livianas). En algunas aproximaciones la formulación de optimización sólo significa aplicar técnicas de optimización a un Search Problem. En este caso, lo que se minimiza es la llamada distancia de viabilidad (*distance to feasibility*). Aún cuando el problema es un problema de optimización real, la distancia de viabilidad se debe incluir en la función objetivo. Esto se realiza para facilitar la navegación del espacio de búsqueda (o *search space*).

En ambos casos (búsqueda y optimización), definimos el problema subyacente, que es el problema de decidir si existe una solución, en el caso del *search problem*, y el problema de decidir si existe una solución con un valor dado de la función objetivo, en el caso de un *optimization problem*.

2.4 Técnicas de resolución del Timetabling Problem

El problema de Timetabling ha sido recientemente estudiado con técnicas pertenecientes a la Inteligencia Artificial [Schaerf,1995], a continuación mostraremos cuales son esas técnicas.

Heurísticas Directas

La mayoría de las técnicas iniciales del School Timetabling se basaban en una simulación de la forma humana de resolver el problema. Tales técnicas se llaman *direct heuristics* (heurísticas directas), y funcionan así: un horario parcial se va extendiendo lección por lección, hasta que a todas las lecciones se las ha asignado a un horario. La idea subyacente de estas aproximaciones es “asignar las materias más restringidas primero”, y ellas difieren solamente en el significado que le dan a la expresión “más restringida”.

Un ejemplo de este método es el sistema SCHOLA de Uhlemann (1969), que se describe en [Junginger,1986]. El sistema se basa en las siguientes tres estrategias:

> A: Asignar la lección más urgente al timeslot más favorable para esa lección.
>
> B: Cuando un timeslot puede usarse solamente para una lección, asignar ese timeslot a esa lección.
>
> C: Mover una lección ya asignada a un timeslot libre para dejar ese timeslot para la lección que queremos asignar actualmente.

Una lección es “urgente” cuando está muy restringida, esto es, cuando el profesor (y la materia) tiene poca disponibilidad y mucha carga horaria. Un timeslot es “favorable” cuando son pocas las lecciones que se pueden asignar allí basandose en la disponibilidad de los demás profesores y cursos.

El sistema SCHOLA asigna lecciones alternando las Estrategias A y B tanto como sea posible. Cuando no se pueden asignar mas lecciones en esta forma, comienza a usar la Estrategia C.

La Estrategia A es la estrategia núcleo, y se emplea en casi todos los sistemas, con diferentes formas de definir “urgencia” y “favorable”. El uso de la Estrategia B puede prevenir que la Estrategia A entre en ciclos sin salidas. La Estrategia C provee una forma limitada de Backtracking para recuperarse de los “errores” de la Estrategia A.

Lógica de Enunciados y Satisfactibilidad

El problema de Satisfactibilidad Proposicional o Booleano (SAT) es el problema de determinar la existencia de una asignación de valores para una fórmula proposicional σ o probar que ésta contiene una contradicción.

Como el problema SAT es fundamental para muchos problemas prácticos en Matemáticas, Ciencia de la Computación e Ingeniería, se desea obtener métodos que puedan resolver la mayor cantidad y tamaño de subconjuntos de problemas SAT.

Una fórmula contiene variables booleanas x_i y conectivos lógicos ∧ (AND), ∨ (OR) y ¬ (NOT). Un literal es una variable o la negación de una variable x_i. Si existe una asignación de valores para una fórmula, la fórmula es *satisfacible* ; de otra manera se llama *insatisfacible* .

Una fórmula en lógica proposicional: es una cadena bien formada que puede contener: [Hamilton, 1981]

- variables proposicionales $x_1, x_2, \ldots, x_n$;
- valores de verdad: T (True), F (False);
- operadores ¬ (not), ∨ (or), ∧ (and);
- paréntesis (para anidar subfórmulas).

Definición: Modelo (o asignación satisfactoria) de una fórmula S es una asignación de valores de verdad a las variables en S que hacen que S sea Verdadera.

Definición: La fórmula S es satisfactible si y solo si existe al menos un Modelo de S. Es insatisfactible en caso contrario.

Una fórmula en CNF (*Conjunctive Normal Form*, Forma Normal Conjuntiva) es una conjunción de un número de cláusulas, donde una cláusula es una disjunción de literales. Se puede ver como

$$\bigwedge_{i=1}^{m} \bigvee_{j=1}^{k(i)} l_{ij} = (l_{11} \vee \ldots \vee l_{1k(1)}) \quad \ldots \wedge \quad (l_{m1} \vee \ldots \vee l_{mk(m)})$$

Ejemplo sencillo:

$$S := \wedge (\neg x2 \vee x1) \wedge (\neg x1 \vee \neg x2 \vee \neg x3) \wedge (x1 \vee x2) \wedge (\neg x4 \vee x3) \wedge (\neg x5 \vee x3)$$

- F está en FNC.
- Es F satisfactible ?

Sí, por ejemplo x_1 y x_2 Verdaderas y x_3, x_4 y x_5 Falsas constituyen un modelo de S (es decir, un renglón de la tabla de verdad que da V).

En el trabajo realizado por [Kenneth-Chin Fat, 2004] se presenta un método para construir Timetables para una Escuela Secundaria de Rotterdam, Holanda. El mismo permite transformar los requerimientos de la Escuela como una Fórmula F en lógica proposicional luego utilizar un Solver SAT para determinar si la fórmula F es o no satisfactible, en caso afirmativo obtiene entonces la asignación horaria para la Escuela.

Reducción a la coloración del grafo

Neufeld y Tartar (1974) [Neufeld-Tartar, 1974] propusieron una reducción del Timetabling Problem al problema de la coloración del grafo. Cada lección se asocia con un vértice en el grafo, y hay una arista entre cada par de lecciones que no pueden asignarse al mismo timeslot a la vez. En particular, las lecciones que comparten profesor o comparten curso (o ambas) se unen mediante una arista en el grafo.

Algoritmos de coloración

Una heurística simple de coloración se llama SEQ. En [Johnson-Aragon,1991] aparece descripta como: se ordenan los vértices v_1, ..., v_n y los colores c_1, ..., c_k. Inicialmente el color c_1 se asigna al vértice v_1. Luego al vértice v_i en su turno se le asigna el color "más chico" que no tiene vértices adyacentes a v_i.

Otra idea levemente diferente se usa en el algoritmo DSATUR de [Brélaz 1979]: en cada paso DSATUR elige el vértice que va a colorear tomando aquel que es adyacente al mayor número de vértices coloreados distinto.

Algoritmos Genéticos

Durante los últimos treinta años ha habido un creciente interés en sistemas basados en los principios de la evolución y la herencia. Tales sistemas mantienen un conjunto o "población" de soluciones posibles.

La población (population) se conforma por un número de individuos, o cromosomas (chromosomes).

Cada individuo representa una solución potencial al problema

Los sistemas evolutivos tienen un proceso de selección de cromosomas basado en una función fitness. También comparten un principio: una población de individuos recibe transformaciones y durante el proceso evolutivo que los transforma, los individuos "luchan" por sobrevivir.

La función fitness da una medida del grado de optimización de un

cromosoma. Cada cromosoma tiene un factor de hard fitness que representa el número de restricciones duras (hard constraints) insatisfechas.

Usamos Programas Evolutivos (EP, Evolution Programs), para referirnos a todos los sistemas basados en evolución. La estructura general de un programa evolutivo es: [Lee, 2000]

```
procedure evolution
begin
        t <- 0
        initialize P(t)
        evaluate P(t)
        while (not termination-condition) do
        begin
                t <- t+1
                select P(t) from P(t-1)
                alter P(t)
                evaluate P(t)
        end
end
```

El programa evolutivo es un algoritmo probabilístico que mantiene una población de "individuos", $P(t)=\{x_1^t, ..., x_n^t\}$ en cada iteración t. En cualquier programa evolutivo el individuo se representa con alguna estructura de datos, digamos S.

A cada solución x_i^t se le aplica la función "fitness". Entonces, se forma una nueva población (iteración t+1) seleccionando los individuos más adecuados (paso de selección). Algunos miembros de la nueva población p(t+1) sufren transformaciones (paso de alteración) mediante operadores "genéticos", para formar nuevas soluciones. Existen transformaciones unarias m_i (tipos mutación), que crean nuevos individuos mediante un pequeño cambio a un único individuo (m_i : S -> S) , y transformaciones de alto orden c_j (tipos crossover), que crean individuos nuevos combinando partes de algunos (dos o varios) individuos (c_j : S x ... x S -> S) .

Luego de un número de generaciones, el programa siempre converge, es decir, se acerca al valor óptimo de la solución.

Los mejores individuos representan una solución razonable, cerca de la óptima.

El método de evolución (evolution method) es la función principal de la población. Aquél define la estrategia de generación de una nueva población a partir de una anterior, usando mutación, cruza (crossover) o propagación simple de los cromosomas viejos. Usualmente, la nueva población se genera iterativamente (los nuevos cromosomas se generan uno a uno a partir de los anteriores).

Claramente, se pueden formular varios programas evolutivos para un problema dado. Pueden utilizar diferentes estructuras de datos para representar un individuo, pueden tener distintos operadores “genéticos” para transformar a los individuos, diferentes métodos para crear una población inicial, o diferentes métodos para manejar las restricciones y los parámetros del problema (tamaño de la población, probabilidades de aplicar operadores diferentes, etc).

Un algoritmo genético ha sido aplicado al problema del School Timetabling en [Colorni,1992]. En ese trabajo se considera el problema de optimización utilizando la función objetivo basada en los siguientes costos (con peso decreciente):

- el costo didáctico: por ej. esparcir las lecciones a lo largo de toda la semana;
- el costo organizacional: por ej tener un profesor disponible por posibles “huecos” temporarios de clases.
- el costo personal: por ej un dia de descanso para cada profesor.

El algoritmo incluye en su espacio de búsqueda los timetables no correctos. La función objetivo incluye el número de violaciones de la restricciones. Para dirigir la búsqueda hacia timetables correctos, a estas violaciones se les da un peso muy alto en la función objetivo.

El algoritmo incluye una fase de búsqueda local que mueve una solución a su mínimo local.

Un algoritmo para resolución del University Timetabling es propuesto por [Molina, 2007]. Utiliza esta técnica metaheurística para solucionar el problema de timetabling de la Universidad de Valparaíso.

Recocido simulado (*Simulated Annealing*)

Es una técnica que hace uso de conceptos originalmente descritos por la mecánica estadística.

Tiene su base en el proceso físico de recocido, el cual primero reblandece un sólido mediante su calentamiento a una temperatura elevada, y luego va enfriando lentamente hasta que las partículas se van posicionando por sí mismas en el "estado fundamental" del sólido. Este proceso tiene su equivalente en el mundo computacional (ver tabla 2.1)

Termodinámica	***Optimización***
Configuración	Solución Factible
Configuración fundamental	Solución Optima
Energía de la configuración	Costo de la Solución
Temperatura	Parámetro de control

Tabla 2.1

El algoritmo selecciona aleatoriamente un candidato de entre los que componen el entorno de la solución actual. Si el candidato es mejor que ella, en términos del criterio de evaluación, entonces es aceptado como solución actual; en caso contrario, será aceptado con una probabilidad que decrece, según crezca la diferencia entre los costos de la solución candidata y actual; cuando el candidato no es aceptado, el algoritmo selecciona otro candidato y se repite el proceso.

El recocido simulado es capaz de encontrar la solución óptima, sin embargo, esta puede alcanzarse tras un número infinito de pasos, en el peor de los casos. La aplicación del método a problemas combinatorios fué propuesto por Kirkpatrick [Kirkpatrick, 1983].

Abramson [Abramson, 1991] aplicó Simulated Annealing al problema de School Timetabling. Los tamaños de los problemas utilizados son de 15 a 101 clases, 100 a 757 materias, 15 a 37 profesores y 15 a 24 aulas.

Búsqueda Tabú (*Tabu Search*)

La búsqueda tabú [Glover, 1993], es un tipo de búsqueda por entornos, que permite moverse a una solución aunque no sea tan buena como la actual. De este modo se puede escapar de óptimos locales y continuar la búsqueda de soluciones aún mejores. La forma de evitar viejos óptimos locales es clasificando un determinado número de los más recientes movimientos como “movimientos tabú”, los cuales no son posibles repetir durante un determinado horizonte temporal. Por lo tanto, en este caso, el escape de los óptimos locales se produce de forma sistemática y no aleatoria.

La búsqueda tabú, al contrario que *simulated annealing* enfatiza procedimientos determinísticos en lugar de aleatorios. Por ello es importante considerar que éstos requieren de la exploración de un gran número de soluciones en poco tiempo. La reducción del esfuerzo computacional es crítica.

Una implementación para Timetabling Problem utilizando *Tabu Search* es la de [Costa, 1994]. Costa empleó en sus datos 32 materias, 780 clases, 65 profesores y 12 aulas.

Schaerf [Schaerf, 1995] utilizó en sus datos 38 materias y 27 a 29 clases por cursos. Ambos aplicaron *Tabu Search* a un problema *School Timetabling* utilizando datos reales de las escuelas de Porrentrug, Suiza y Potenza, Italia respectivamente.

Capítulo 3 – Modelación del Problema

3.1 Introducción

Esta sección muestra cómo arma los horarios el encargado de la escuela que tomamos como caso de estudio.

Presentamos los datos obtenidos de la escuela. Lo tomamos de una planilla (ver sección de Anexos) del horario año 2006 armada por la persona encargada de la escuela.

Con esta información llenamos las estructuras de datos que definimos en la primer Fase del Plan de Pruebas de la Seccion 7.4 que servirán de entrada a los algoritmos que se estudian.

3.2 Caso de estudio

En una entrevista realizada con la persona encargada de armar el Horario en el Colegio Secundario de nuestro caso, explicó que antes de la terminación del año, los profesores deben entregar una lista de bloques en los que están disponibles para el dictado de las materias que les corresponden. Esta información es firmada bajo declaración jurada, y debe superar la cantidad de horas que tiene asignadas el profesor en relación a las materias y cursos en los que dicta.

Un ejemplo

Prof. Terzaghi Hrs. Martes 4ta y 5ta, Miercoles 1ra y 2da, Jueves 2da y 4t, Viernes 1ra y 3ra.

Lo que totaliza 8 hrs semanales. Y se distribuyen en 4 horas para Sociales del curso 7A y 4 horas en la misma materia del curso 7B.

Cuando todos los profesores titulares entregan el detalle de horas al colegio, éste

verifica la veracidad de los mismos y luego se lo entrega a la persona encargada del armado.

El tiempo que le lleva armar un Horario “aceptable”, es decir, que cumpla con la mayoría de los requisitos de los profesores es medido en meses, para esta persona responsable generalmente todo el período de receso escolar, esto es, 3 meses.

La misma persona utiliza para la resolución una forma “lineal” en el orden que realiza la asignación; para esto toma de la lista de profesores-materias, el que tiene más cantidad de horas y le asigna el bloque en el Horario; así sigue de a un profesor-materia.

Cuando encuentra un bloque que no puede asignar, generalmente deshace la última asignación e intenta con el nuevo profesor. Esto puede llevar a deshacer totalmente el Horario semi-armado y comenzar nuevamente quizás cambiando el orden, pero esto no asegura un resultado exitoso.

Generalmente queda armado un Horario con “huecos” y se asignan esos bloques a profesores que no las pidieron o lo peor que puede pasar es dejarlos libres, sin dictado de lecciones.

Entre las restricciones “obligatorias” que debe manejar el encargado son:

- Una materia no se puede dictar en dos bloques intercalados para el mismo curso y dia. Por ej Lengua-Matematicas-Lengua.
- Hay una cantidad máxima de bloques por dia que se puede dictar una materia a un curso. Cualquiera de las materias Lengua, Matemáticas, etc NO se dictan más de 2 bloques por día y curso.
- Si quedan “huecos” en el Horario, deben quedar en los horarios extremos del dia, sería la 1era o la última hora.

También hay restricciones “opcionales” como:

- Si un profesor debe viajar desde lejos se busca asignarle todas las horas seguidas para los días que dicta.
- Se busca que las materias “pesadas” como Historia NO se dicten en la primer hora de la mañana.

Pero las restricciones más complejas son:

- Hay materias que necesitan lugares especiales por ejemplo Gimnasia, que se dictan en lugares alejados del edificio prinicipal, lo que lleva un bloque especial que puede ser fuera del Horario normal.
- Hay materias de diferentes cursos que se “juntan” para dictarse en el mismo bloque y con uno o ambos profesores. El caso es Ingles de 7A y 7B que se dicta en conjunto por ambos profesores en un aula con la capacidad suficiente para ambos cursos. Otro caso es el de Gimnasia que se dicta a todos los cursos a la vez, pero separados los varones de las mujeres en distinto horario.
- Otro caso que se da habitualmente es cuando un profesor pide licencia por algún motivo, que se debe buscar Profesores Suplentes que puedan dictar en los mismos bloques horarios que ya tenía asignado el profesor titular. Ha pasado que cuando ocurre esto la materia NO se dicta porque los profesores suplentes no pueden dictar en esas horas. Es preferible esta solución a tener que reorganizar el Horario semanal.

Por todas estas restricciones es que el encargado lleva una “calificación” de cada profesor para saber con cuales comenzar la asignación. Esta “calificación” se basa en factores como antigüedad, cantidad de materias y horas, responsabilidad, si dicta o no en varios colegios, flexibilidad en cuanto a posiblidad de cambio de horas asignadas y libres, y otras “ideas” sobre el profesor para asegurar la consistencia del organigrama.

Siguiendo con este último punto el encargado nos comentó que la dificultad para asignar una materia con muchas horas semanales se da también para las que tiene pocas horas disponibles por el profesor.

3.3 Construcción manual de un Timetable

La escuela divide a todos los alumnos en diferentes cursos: 7mo, 8vo y 9no. A su vez, cada curso se subdivide en aulas que se identifican con una letra A ó B.

Todos las clases se dictan dentro de 5 bloques diarios que llamaremos *Timeslots*, donde cada bloque tiene un horario fijo.

Timeslot	Día / hora
1	Lun 07-08 AM
2	Lun 08-09 AM
3	Mar 07-08 AM
....	.....
24	Vie 11 AM – 12 AM
25	Vie 12 AM- 01 PM

El proceso de construcción manual de un *Timetable* en la escuela involucra las siguientes tareas:

1. Identificar las materias (o cursos) que se dictan.
2. Asignar un profesor a cada materia de cada curso.
3. Establecer los requerimientos de cada materia / curso, especificando cantidad de horas necesarias.
4. Asignar cada *timeslot* de acuerdo a la política de alocación, preferencias y restricciones que deben cumplirse.

A continuación nos concentramos en estas tareas:

1. Identificar las materias (o cursos) que se dictan

Materias dictadas

ID	Nombre Mat
1)	Sociales
2)	Naturales
3)	Lengua
4)	Artística
5)	Inglés
6)	Matemática
7)	Computación
8)	Catequesis

Cursos

ID	Nombre	Nombre corto
1)	7A	K1
2)	7B	K2
3)	8A	K3
4)	8B	K4
5)	9A	K5
6)	9B	K6

2. Asignar un profesor a cada materia de cada curso

Armamos los pares materia-profesor, junto con las preferencias horarias del profesor.

ID	Nombre Mat- Nombre Prof.	Timeslots (1=puede, 0=no puede) 1,2,3,4,5,6,7,.....,24,25
1)	Sociales – Terzaghi	1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,1,1,0
2)	Sociales – Lorenzo	1,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,0,0
3)	Sociales – Inchausti	1,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,0,0
4)	Naturales – Manuele	1,1,1,1,0,1,1,1,0,0,1,1,1,0,0,1,1,1,0,0,1,1,1,0,0
5)	Naturales – Diaz	0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,1,1,1,1,0
6)	Lengua – Lonne	1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,0,0,1,1,1,0,1,1,1,1
7)	Lengua – Pintos	0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0
8)	Artística – Solas	1,0,0,1,0,1,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,1,0,1,0
9)	Artística – Reyes	0,0,0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0
10)	Inglés – Clark	0,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0
11)	Inglés – Sisto	0,1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0
12)	Inglés – Sarsfield	0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0
13)	Matemática – Warning	0,0,0,1,1,1,1,1,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0
14)	Matemática – Bruno	0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,1,1,0,1,0,1,1,0,0,0
15)	Matemática – Valori	0,0,0,0,0,1,1,1,1,0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0
16)	Computación – Cuenca	0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0
17)	Computación – Aguerre	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1
18)	Catequesis – Ferreyro	0,0,0,0,0,0,0,1,1,1,0,0,0,1,1,0,0,1,0,1,0,0,0,0,1
19)	Catequesis – Dambrosio	0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1

Aquí podemos ver por ejemplo que el profesor Terzaghi que dicta la materia Sociales con ID = 1, puede acudir a la escuela sólo en los timeslots 1 y 2 (Lunes primer y segundo período), del 16 al 19 (Jueves del primer al cuarto período) y del 23 al 24 (Viernes tercer y cuarto período).

En forma más gráfica tenemos las preferencias horarias del profesor:

Prof. Terzaghi	***Lunes***	***Martes***	***Miércoles***	***Jueves***	***Viernes***
1ra.					
2da.					
3ra.					
4ta.					
5ta.					

3. Establecer los requerimientos de cada materia / curso, especificando cantidad de horas necesarias

Aquí se establece para cada Materia de cada Profesor, cuántas horas va a dictar esa materia en los Cursos del colegio.

Tuplas de Materias – Profesor y Curso:

ID	Nombre Mat- Nombre Prof.	Nombre Curso	Cantidad Hrs a dictar	Nombre corto
1)	Sociales – Terzaghi	7A	4	M1
2)	Sociales – Terzaghi	7B	4	M2
3)	Sociales – Lorenzo	8B	4	M3
4)	Sociales – Lorenzo	9A	4	M4
5)	Sociales – Inchausti	8A	4	M5
6)	Sociales – Inchausti	9B	4	M6
7)	Naturales – Manuele	7A	4	M7
8)	Naturales – Manuele	7B	4	M8
9)	Naturales – Manuele	8A	4	M9
10)	Naturales – Manuele	8B	4	M10
11)	Naturales – Diaz	9A	4	M11
12)	Naturales – Diaz	9B	4	M12

13)	Lengua – Lonne	7A	4	M13
14)	Lengua – Lonne	8A	4	M14
15)	Lengua – Lonne	8B	4	M15
16)	Lengua – Lonne	9A	4	M16
17)	Lengua – Lonne	9B	4	M17
18)	Lengua – Pintos	7B	4	M18
19)	Artística – Solas	7A	2	M19
20)	Artística – Solas	7B	2	M20
21)	Artística – Solas	9A	2	M21
22)	Artística – Solas	9B	2	M22
23)	Artística – Reyes	8A	2	M23
24)	Artística – Reyes	8B	2	M24
25)	Inglés – Clark	7A	3	M25
26)	Inglés – Clark	8A	3	M26
27)	Inglés – Clark	9A	3	M27
28)	Inglés – Sisto	8B	3	M28
29)	Inglés – Sisto	9B	3	M29
30)	Inglés – Sarsfield	8B	3	M30
31)	Matemática – Warning	7A	4	M31
32)	Matemática – Warning	7B	4	M32
33)	Matemática – Bruno	8A	4	M33
34)	Matemática – Bruno	8B	4	M34
35)	Matemática – Valori	9A	4	M35
36)	Matemática – Valori	9B	4	M36
37)	Computación – Cuenca	7B	1	M37
38)	Computación – Cuenca	9B	1	M38
39)	Computación – Aguerre	8A	1	M39
40)	Computación – Aguerre	8B	1	M40
41)	Computación – Aguerre	9A	1	M41
42)	Catequésis – Ferreyro	8A	2	M42
43)	Catequésis – Ferreyro	8B	2	M43
44)	Catequésis – Ferreyro	9A	2	M44
45)	Catequésis – Ferreyro	9B	2	M45
46)	Catequésis – Dambrosio	7A	2	M46
47)	Catequésis – Dambrosio	7B	2	M47

Clases

El término Clase fué introducido en la Sección 2.3 de este trabajo, aquí daremos una ampliación al mismo ya que es utilizado como base en los subsiguientes capítulos de esta Tesis.

Una clase i representa a una tupla conformada por un número de lección X de un par materia-profesor Y correspondiente a un curso Z.

Un ejemplo de clase:

Clase con ID igual a 1 corresponde a la primer lección que dicta el profesor Terzaghi de la materia Sociales al curso 7A.

Lista de Clases que son dictadas en la escuela del caso de estudio, siendo una lección una hora cátedra de dictado.

ID	Mat-Prof-Curso	Nro de lección
1	M1	1
2	M1	2
3	M1	3
4	M1	4
5	M2	1
6	M2	2
7	M2	3
8	M2	4
9	M3	1
....	....	...
...	...	...
140	M46	1
141	M46	2
142	M47	1
143	M47	2

Vemos que dividiendo cada materia en sus correspondientes cantidad de horas /lecciones dictadas, nos quedan 143 Clases, las cuales se utilizarán en la siguiente etapa para asignarseles los bloques disponibles del Timetable.

4. Asignar cada *timeslot* de acuerdo a la política de alocación, preferencias y restricciones que deben cumplirse.

El encargado de la escuela utiliza para la resolución una forma “lineal” en el orden que realiza la asignación; para esto toma de la lista de clases, la que tiene más cantidad de horas y le asigna el período en el *Timetable y* así sigue de a una clase.

Por ejemplo, puede comenzar con las 4 clases Sociales – Terzaghi que se dictan al curso 7A, entonces asigna cuatro *timeslots* a estas clases.

Aquí hay que tener en cuenta dos posibilidades:

A. No Utilizar la restricción Con Preferencias: Asignar los *timeslots* que estén disponibles en un orden predeterminado, por ejemplo primero los del dia Lunes, luego Martes y así siguiendo.

B. Utilizar la restricción Con Preferencias: Asignar los *timeslots* disponibles pero utilizar sólo aquellos en los cuales el profesor identificó como preferidos.

Timetable Parcial (Sin preferencias)

Curso 7A

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1	Sociales - Terzaghi				
Hora 2	Sociales - Terzaghi				
Hora 3	Sociales - Terzaghi				
Hora 4	Sociales - Terzaghi				
Hora 5					

Timetable Parcial (Con preferencias)

Curso 7A

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1				Sociales - Terzaghi	
Hora 2				Sociales - Terzaghi	
Hora 3					Sociales - Terzaghi
Hora 4					Sociales - Terzaghi
Hora 5					

Luego continúa con las 4 clases Sociales – Terzaghi para el curso 7B, el *timetable* parcial se ve:

Timetable Parcial (Sin preferencias)

Curso 7A

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1	Sociales - Terzaghi				
Hora 2	Sociales - Terzaghi				
Hora 3	Sociales - Terzaghi				
Hora 4	Sociales - Terzaghi				
Hora 5					

Curso 7B

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1		Sociales - Terzaghi			
Hora 2		Sociales - Terzaghi			
Hora 3		Sociales - Terzaghi			
Hora 4					
Hora 5	Sociales - Terzaghi				

Timetable Parcial (Con preferencias)

Curso 7A

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1				Sociales - Terzaghi	
Hora 2				Sociales - Terzaghi	
Hora 3					Sociales - Terzaghi
Hora 4					Sociales - Terzaghi
Hora 5					

Curso 7B

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1	Sociales - Terzaghi				
Hora 2	Sociales - Terzaghi				
Hora 3				Sociales - Terzaghi	
Hora 4				Sociales - Terzaghi	
Hora 5					

Cuando el empleado encuentra un *timeslot* que no puede asignar, generalmente deshace la última asignación e intenta con el nuevo profesor. Esto puede llevar al empleado a deshacer totalmente el *timetable* y comenzar nuevamente quizás cambiando el orden (un "*Backtracking* manual"). Tal decisión no asegura un resultado exitoso.

A continuación vemos el *timetable* resultante de aplicar el método manual:

Timetable Completo (Con Preferencias)

Curso 7A

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1		Artistica - Solas	Naturales - Manuele	Sociales - Terzaghi	Artistica - Solas
Hora 2	Naturales - Manuele	Lengua - Lonne	Naturales - Manuele	Sociales - Terzaghi	Lengua - Lonne
Hora 3	Lengua - Lonne	Matematicas - Warning	Catequesis - Dambrosio	Naturales - Manuele	Sociales - Terzaghi
Hora 4	Matematicas - Warning	Ingles - Clark	Matematicas - Warning	Lengua - Lonne	Sociales - Terzaghi
Hora 5		Ingles - Clark	Matematicas - Warning	Ingles - Clark	Catequesis - Dambrosio

Curso 7B

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1	Sociales - Terzaghi	Matematicas - Warning	Lengua - Pintos	Lengua - Pintos	Computacion - Cuenca
Hora 2	Sociales - Terzaghi	Matematicas - Warning	Lengua - Pintos	Lengua - Pintos	Artistica - Solas
Hora 3	Naturales - Manuele	Naturales - Manuele	Matematicas - Warning	Sociales - Terzaghi	Naturales - Manuele
Hora 4	Naturales - Manuele	Ingles - Sarsfield	Catequesis - Dambrosio	Sociales - Terzaghi	Catequesis - Dambrosio
Hora 5	Matematicas - Warning	Ingles - Sarsfield	Artistica - Solas	Ingles - Sarsfield	

Curso 8A

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1	Naturales - Manuele	Naturales - Manuele	Lengua - Lonne	Matematicas - Bruno	Matematicas - Bruno
Hora 2	Lengua - Lonne	Naturales - Manuele	Artistica - Reyes	Matematicas - Bruno	Matematicas - Bruno
Hora 3	Sociales - Inchausti	Artistica - Reyes	Naturales - Manuele	Catequesis - Ferreyro	Lengua - Lonne
Hora 4	Ingles - Clark	Lengua - Lonne	Matematicas - Warning	Computacion - Aguerre	Sociales - Inchausti
Hora 5	Ingles - Clark		Catequesis - Ferreyro	Sociales - Inchausti	Sociales - Inchausti

Curso 8B

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1	Lengua - Lonne	Matematicas - Bruno	Artistica - Reyes	Naturales - Manuele	Naturales - Manuele
Hora 2	Sociales - Lorenzo	Matematicas - Bruno	Sociales - Lorenzo	Naturales - Manuele	Naturales - Manuele
Hora 3	Sociales - Lorenzo	Catequesis - Ferreyro	Matematicas - Bruno	Lengua - Lonne	Sociales - Lorenzo
Hora 4	Ingles - Sisto	Artistica - Reyes	Ingles - Sisto	Matematicas - Bruno	Lengua - Lonne
Hora 5	Ingles - Sisto	Lengua - Lonne		Catequesis - Ferreyro	Computacion - Aguerre

Curso 9A

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1	Sociales - Lorenzo	Matematicas - Valori	Sociales - Lorenzo	Matematicas - Valori	Sociales - Lorenzo
Hora 2	Ingles - Clark	Matematicas - Valori	Lengua - Lonne	Matematicas - Valori	Sociales - Lorenzo
Hora 3	Ingles - Clark	Lengua - Lonne	Artistica - Solas	Naturales - Diaz	Naturales - Diaz
Hora 4	Artistica - Solas	Catequesis - Ferreyro	Catequesis - Ferreyro	Naturales - Diaz	Naturales - Diaz
Hora 5	Lengua - Lonne		Matematicas - Warning	Computacion - Aguerre	Lengua - Lonne

Curso 9B

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
Hora 1	Artistica - Solas		Matematicas - Valori	Naturales - Diaz	Naturales - Diaz
Hora 2	Ingles - Sisto	Sociales - Inchausti	Matematicas - Valori	Naturales - Diaz	Naturales - Diaz
Hora 3	Ingles - Sisto	Matematicas - Valori	Lengua - Lonne	Sociales - Inchausti	Sociales - Inchausti
Hora 4	Lengua - Lonne	Matematicas - Valori	Lengua - Lonne	Sociales - Inchausti	Artistica - Solas
Hora 5	Computacion - Cuenca	Catequesis - Ferreyro	Ingles - Sisto	Lengua - Lonne	Catequesis - Ferreyro

Generalmente queda armado un *timetable* con “huecos” y se asignan esas horas a profesores que no las pidieron. También pueden quedar libres, sin dictado de clases.

El método manual se puede concebir como dos estrategias:

1)Intentar de a un *timeslot* hasta que se completa el *Timetable* o por lo menos se llena en forma satisfactoria un porcentaje de la carga horaria. Y volviendo atrás cuando no se puede continuar.

2)Asignar en forma satisfactoria un porcentaje y luego hacer una segunda, tercera, y más pasadas corrigiendo y reasignando horas hasta llegar a un *timetable* con la mayor cantidad de restricciones validadas.

3.4 Definiciones

Aquí realizamos un compendio de definiciones encontradas en la bibliografía citada en el Capítulo 2 del Estado del Arte en Timetabling, que nos serán útiles para representar de forma unificada la información recopilada en la Escuela Secundaria. La representación será igual (o similar) para todos los algoritmos comparados en los siguientes capítulos de este trabajo.

Primero definimos índices que representan los rangos de valores mínimos y máximos para cada dato que maneja el *Timetable*.

k : Curso del colegio	con 1 <= k <= K
i : Materia-Profesor	con 1 <= i <= D
j : Número de lección de una materia	con 1 <= j <= U
d : Día	con 1 <= d <= S
t : Hora	con 1 <= t <= T

Siendo K el máximo número de cursos del colegio; D es la cantidad de materias máximo sumando todos los cursos; U es el número máximo de veces que se dicta una materia semanalmente. Por último, S es la cantidad de días de la semana y T la cantidad de horas diarias en las que se dictan clases.

Cada índice está en el rango de 1 a su máximo, lo que nos da un total de **K x D x U x S x T** Clases.

Esto nos da el valor máximo de combinaciones posibles de Materias, Cursos y Lecciones a asignar en los Días y Horas que se divide la semana para el dictado de las Clases a los alumnos de la Escuela.

Currículum

Un currículum [Kenneth-Chin Fat, 2004] es una estructura de datos que nos permite volcar la carga horaria de cada materia para cada curso.

Es una matriz cuyas filas representan los cursos y las columnas representan los pares materia-profesor. El elemento (k,i) del currículum contiene el número de lecciones dictadas en el curso k para el par materia-profesor i.

Armamos el currículum de los cursos:

	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***	***19***
7A	4	0	0	4	0	4	0	2	0	3	0	0	4	0	0	0	0	0	2
7B	4	0	0	4	0	0	4	2	0	0	0	3	4	0	0	1	0	0	2
8A	0	0	4	4	0	4	0	0	2	3	0	0	0	4	0	0	1	2	0
8B	0	4	0	4	0	4	0	0	2	0	3	0	0	4	0	0	1	2	0
9A	0	4	0	0	4	4	0	2	0	3	0	0	0	0	4	0	1	2	0
9B	0	0	4	0	4	4	0	2	0	0	3	0	0	0	4	1	0	2	0

Esta información nos será útil en el Capítulo 7 como datos de entrada a los algoritmos que probaremos.

Matriz de alocación

Cada curso tendrá una matriz de alocación [Kenneth-Chin Fat, 2004].

La matriz de alocación de un curso k Ak(i,j) se construye como sigue. Las filas representan todos los pares materia-profesor, que se dictan en ese curso (o no). Si un par materia-profesor dicta n lecciones en el curso k, los primeros n elementos de la fila correspondiente a Ak (i,j) contendrá unos.

Cada columna representa en número de lección, por lo que la primer lección del materia-profesor i' del curso k se representa por Ak (i',1) y la segunda lección por Ak (i',2).

La descripción formal de la matriz de alocación del curso k:

$$Ak(i,j) = \begin{cases} 1, \text{ si materia-profesor i dicta la j-esima lección al curso k;} \\ 0, \text{ si no.} \end{cases} \quad (2.1)$$

Tomando el currículum de más arriba veremos cómo queda la matriz de alocación para los cursos que hay:

k1	j1	j2	j3	j4
1	1	1	1	1
2	0	0	0	0
3	0	0	0	0
4	1	1	1	1
5	0	0	0	0
6	1	1	1	1
7	0	0	0	0
8	1	1	0	0
9	0	0	0	0
10	1	1	1	0
11	0	0	0	0
12	0	0	0	0
13	1	1	1	1
14	0	0	0	0
15	0	0	0	0
16	0	0	0	0
17	0	0	0	0
18	0	0	0	0
19	1	1	0	0

matriz alocación Ak1 del curso k1 (7A)

Para el caso de estudio necesitamos entonces la matriz de alocación Ak1 , Ak2 hasta la Ak6, ya que se dictan seis cursos.

Relaciones SameTeacher y TeacherAvailable

En [Kenneth-Chin Fat, 2004] se definen dos relaciones que nos serán útiles para expresar los requerimientos de los pares materia-profesor y de los horarios disponibles de los profesores, ellas son: *SameTeacher* (igual profesor) y

TeacherAvailable (profesor disponible).

Si por ejemplo queremos comparar dos pares materia-profesor del mismo profesor, solamente elegimos aquellos pares i e i' para los cuales SameTeacher(i,i')=1. Y si queremos determinar si un par materia-profesor i está disponible en un día d y una hora t vemos si TeacherAvailable(i,d,t)=1.

Las definiciones formales de estas relaciones es :

•SameTeacher(i, i')

Sean i e i' dos pares materia-profesor.

$$\text{SameTeacher}(i,i') = \begin{cases} 1, & \text{si la materia-profesor i y i' representan al mismo profesor;} \\ 0, & \text{caso contrario.} \end{cases}$$

•TeacherAvailable(i,d,t)

Sean i un materia-profesor, d un día y t un horario.

$$\text{TeacherAvailable}(i,d,t) = \begin{cases} 1, & \text{si el par materia-profesor i está disponible el día d en el horario t;} \\ 0, & \text{caso contrario.} \end{cases}$$

Para nuestro caso, la primer relación definida queda:

1)SameTeacher(1,1) =1
2)SameTeacher(1,2) =0
3)SameTeacher(1,3) =0
4)SameTeacher(1,4) =0
5) .....
6) SameTeacher(1,19) = 0
7)SameTeacher(2,1) =0
8)SameTeacher(2,2) =1
9)SameTeacher(2,3) =0
10)SameTeacher(2,4) =0
11).....
12) SameTeacher(2,19)=0
13) .....
14)SameTeacher(18,18) = 1
15)SameTeacher(18,19) =0
16)SameTeacher(19,1) =0
17)SameTeacher(19,2) = 0
18) .....
19)SameTeacher(19,18) =0
20)SameTeacher(19,19) =1

Donde dice SameTeacher(1,1)=1 se lee “Verdadera”; o sea que el par materia-profesor 1 (“Sociales – Terzaghi”) y el par 3 (“Sociales - Terzaghi”) comparten el mismo profesor “Terzaghi”.

Donde dice SameTeacher(1,2)=0 significa “Falso”; o sea que el par materia-profesor 1 (“Sociales-Terzaghi”) y el par 2 (“Sociales-Lorenzo”) son dictadas por diferentes profesores, “Terzaghi” y “Lorenzo” .

Las lineas punteadas las colocamos para indicar que hay más combinaciones que no escribimos en la tabla por el espacio que ocupa en la tabla. Por ejemplo en la linea punteada del renglón 5) iría SameTeacher(1,5)=0 o 1 y así siguiendo hasta SameTeacher(1,18) = 0 o 1.

Luego se ve que cuando coinciden el Profesor en un par materia-profesor siempre se asigna un 1 a la relación SameTeacher.

Ahora vemos la asignación de la relación TeacherAvailable, suponiendo que todos los profesores están disponibles para cualquier día y hora de nuestro ejemplo:

- TeacherAvailable(1,1,1) = 1
- TeacherAvailable(1,1,2) = 1
- TeacherAvailable(1,1,3) = 0
- TeacherAvailable(1,1,4) = 0
- TeacherAvailable(1,1,5) = 0
- TeacherAvailable(1,2,1) = 0
- TeacherAvailable(1,2,2) = 0
- .....
- TeacherAvailable(1,5,3) = 1
- TeacherAvailable(1,5,4) = 1
- TeacherAvailable(1,5,5) = 0
- TeacherAvailable(2,1,1) = 1
- TeacherAvailable(2,1,2) = 1
- .....
- TeacherAvailable(19,5,4) = 1
- TeacherAvailable(19,5,5) = 1

Interpretando la relación TeacherAvailable(1,1,1)=1 en lenguaje natural significa que el par materia-profesor 1 (“Sociales-Terzaghi”) está disponible para dictar en el día 1 (“Lunes”) y en la hora 1 (“hora1”); que TeacherAvailable(2,1,2) nos informa que el par 2 (“Sociales-Lorenzo”) está disponible el día 1 (“Lunes”) en la hora 2 (“Hora 2”) y así para todos los 19 pares materia-profesor.

Capítulo 4 – Timetabling utilizando Coloración del grafo

4.1 Introducción

Una aplicación importante de la técnica de coloración del grafo es el problema de *timetabling*.

El problema de determinar el número mínimo (o un número razonable) de *timeslots* necesarios para asignar todas las materias de todos los cursos de acuerdo a las restricciones dadas por la escuela se puede reducir al problema de coloración del grafo.

Diversos textos citan a Welsh y Powell como los descubridores de la similitud o isomorfismo entre el problema de Timetabling y el problema de coloración de vértices de un grafo.

4.2 Definiciones

Un grafo G puede definirse como un conjunto de vértices V(G) = $\{v_1, v_2, ..., v_n\}$ y un conjunto de aristas E(G) = $\{e_1, e_2, ..., e_m\}$. Cada arista de E(G) es un par de vértices de V(G). Para una arista e = {x,y} (que a menudo abreviamos escribiendo "e = xy"), diremos que si xy pertenece a E(G), entonces x e y son *adyacentes* en G.

Dado un grafo G, una (vértice-) coloración de G es una función f con dominio G e imagen C, donde los elementos de C son llamados *colores.* Una k-coloración G es una función f que usa exactamente k colores y satisface la propiedad de que f(x) /= f(y) para x e y adyacentes en G. Decimos entonces que el grafo G es *k-coloreable*. [Chartrand, 1993]

Notemos que si G tiene un ciclo, entonces G no tiene coloración, ya que f(x) = f(x). El número cromático X(G) de G es el mínimo número k tal que exista un k-coloración apropiada de G.

Una mínima coloración de G es una coloración que usa la menor cantidad de

colores posible.

El problema de determinar el número cromático de un grafo es formalmente NP-Completo para grafos generales [Garey, 1979].

Ejemplo:

La figura inferior muestra una 3-coloración del grafo G con cinco vértices y cinco aristas. (Los números representan los 3 colores).

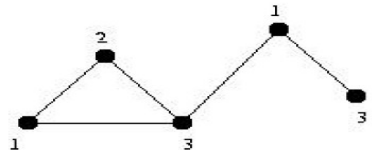

Figura 2.1

4.3 Creación del grafo

Una vez que tenemos los datos del curso, podemos construir un grafo asociado a esos datos del siguiente modo [Redl,2004]: los vértices representan las clases, y las aristas conectan cada par de items “incompatibles” o “conflictivos”.

Un vértice en G representa una Clase, una arista representa el par de clases que entran en conflicto, y un color representa el *timeslot* en el que esa clase particular va a ser alocada.

Las aristas indican los pares de clases que no pueden ser asignados al mismo *timeslot*. Algunos de estos arcos reflejan condiciones esenciales (o duras), estos se agregan para eliminar conflictos del grafo. Otros arcos reflejan condiciones preferenciales y pueden o no agregarse al grafo.

El agregado de aristas se hace como sigue [Redl,2004]:

Si el PROFESOR de la clase c_i y el PROFESOR de la clase c_j son el mismo, entonces debemos agregar un arco entre el vértice v_i y el vértice v_j , ya que esas

materias no puede asignárseles el mismo *timeslot*. Entonces que dos vértices sean adyacentes implica que no puede asignárseles el mismo color.

Si el CURSO de la clase c_i y el CURSO de la clase c_j son el mismo, entonces debemos agregar un arco entre el vértice v_i y el vértice v_j , ya que a esas materias no pueden asignárseles el mismo timeslot.

Clases de ejemplo

1.mat1-le1-K1 / p1
2.mat1-le2-K1 / p1
3.mat2-le1-K1 / p2
4.mat2-le2-K1 / p2
5.mat3-le1-K1 / p1
6.mat3-le2-K1 / p1
7.mat1-le1-K2 / p2
8.mat1-le2-K2 / p2
9.mat2-le1-K2 / p3
10.mat2-le2-K2 / p3
11.mat3-le1-K3 / p4
12.mat3-le2-K3 / p4

("K" significa Curso, "mat" es de Materia, "p" es de Profesor, "le" es de Lección)

La forma de construir el grafo G para el ejemplo es como sigue: tenemos un conjunto de 12 clases $\{c_1, c_2, ..., c_n\}$ n=12, a ser asignadas. Cada c_i se corresponde con un vértice v_i en G. Por lo tanto G contiene n=12 vértices, y $V(G) = \{v_1, v_2, ..., v_n\}$.

Pasamos a agregar las aristas:

– Primero, colocamos las aristas entre vértices que son del mismo profesor: aquí tenemos por ejemplo que el profesor p_1 está en los vértices 1,2, 5 y 6 así que colocamos las aristas correspondientes. Luego vemos el profesor p_2 que dicta en los vértices 3, 4, 7 y 8. Seguimos con el profesor p_3 y finalmente con el p_4.

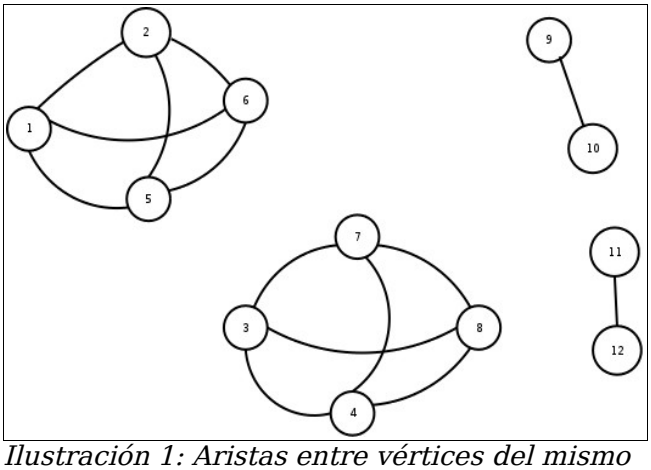

Ilustración 1: Aristas entre vértices del mismo profesor

– Segundo, colocamos aristas entre los vértices que representan materias-profesor de un mismo curso: vemos que los vértices v_1 al v_6 pertenecen al mismo curso K1, por lo que agregamos las correspondientes aristas entre ellos. Los vértices v_7 al v_{10} son del curso K2, así que tiene que agregarse las aristas que unen esos vértices. Finalizamos con los vértices v_{11} y v_{12} que representan materias del curso K3.

El grafo resultante se ve como sigue:

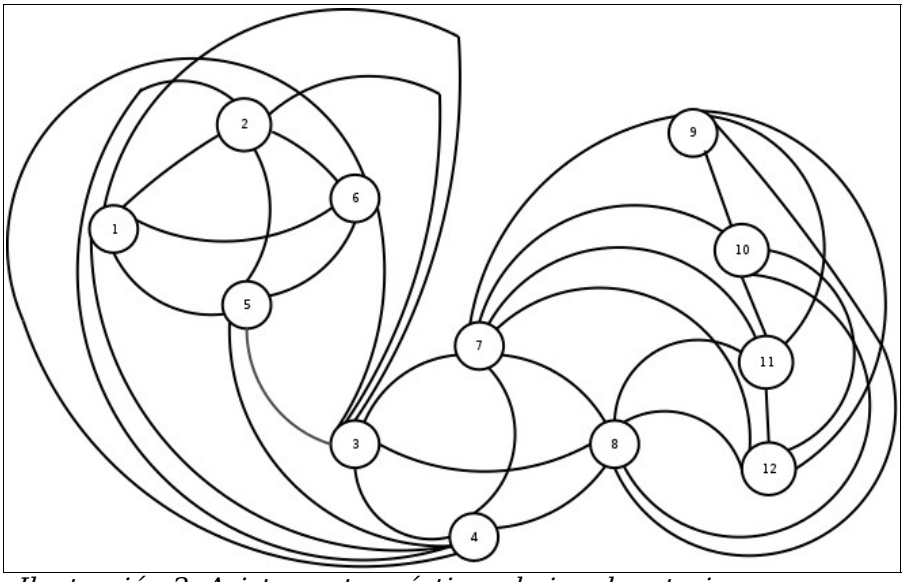

Ilustración 2: Aristas entre vértices de igual materia y curso

Tenemos en el grafo un total de 12 vértices y 34 aristas.

4.4 Coloración del grafo

Una vez que construimos el grafo, que llamaremos grafo de conflictos, podemos intentar colorear los vértices. Luego usaremos esa coloración para construir el *Timetable*.

Recordemos que en una coloración apropiada del grafo, a un par de vértices v_i y v_j se les asignan diferentes colores si existe un arco entre ellos. Si no hay arco entre ellos es claro que pueden o no tener el mismo color.

Hay varias maneras del realizar el coloreado en forma manual. No son complejas y pueden diferir solamente en el orden en que se realiza la selección de los vértices y/o de los colores.

Si numeramos los colores r_1, r_2, ..., r_k en el orden que se crean durante la coloración, entonces tendremos los siguientes colores para utilizar:

r_1 = Dia1-Hora1
r_2 = Dia1-Hora2
r_3 = Dia2-Hora1
r_4 = Dia2-Hora2
r_5 = Dia3-Hora1
r_6 = Dia3-Hora2

Para colorear nuestro grafo en forma manual, naturalmente recorremos los vértices comenzando con el vértice 1 al cual le asignamos el color r_1, luego con el vértice 2 que le asignamos el color r_2 porque es adyacente del vértice 1, y así continuamos hasta el vértice 7; por cada vértice que tomamos recorremos los colores del r_1 al r_6, por lo tanto a este vértice le corresponde el r_1.

Siguiendo con este mecanismo vemos el grafo coloreado:

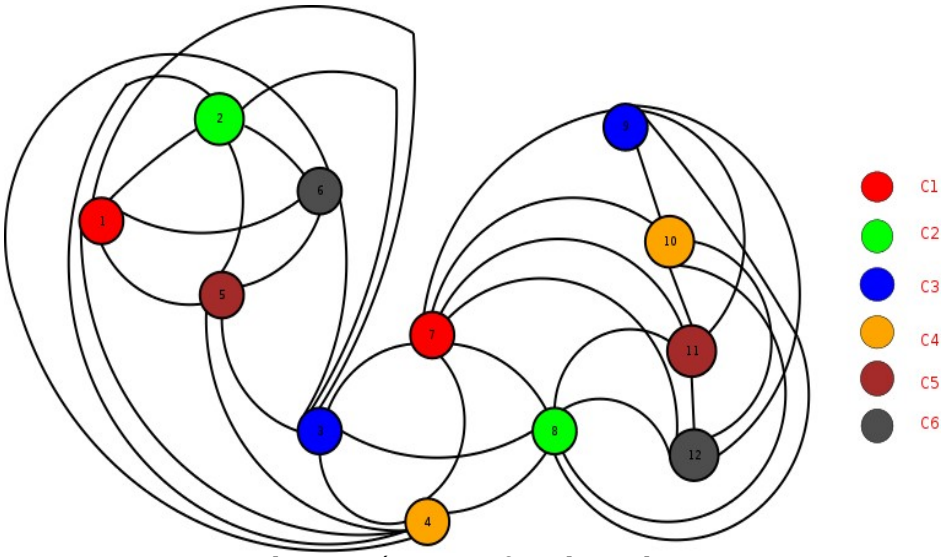

Ilustración 3: Grafo coloreado

A continuación describimos un algoritmo para la coloración del grafo llamado DSATUR presentado por [Brelaz,1979] como solución de la Coloración del Grafo.

Algoritmo DSATUR.

Consideramos un grafo G=(V, E) simple (sin ciclos) y no-dirigido.

El algoritmo DSATUR [Brelaz, 1979] es un algoritmo secuencial de coloración, que toma de a un vértice por vez de forma que:

- al comienzo el grafo no está coloreado
- se colorea un vértice que no está ya coloreado
- DSATUR se detiene cuando todos los vértices de G están coloreados.

Con más detalle el algoritmo es como sigue:

1. Ordena los vértices por orden decreciente de grados (*degree*) de saturación.

2. Colorea el vértice de máximo *degree* con el color 1.
3. Elige un vértice con el máximo DSAT.
4. Colorea este vértice con el color más chico posible.
5. Si todos los vértices están coloreados termina, sino vuelve al paso 3.

La función DSAT (v) calcula el grado de saturación del vértice v de la siguiente manera:

si ningún vecino de v está coloreado **entonces**

DSAT(v) = degree(v)

sino

DSAT(v)= número de colores diferentes usados en los vecinos de v.

En el Capítulo 7 de este trabajo mostramos cómo utilizamos la implementación de J. Culberson [Culberson] del algoritmo DSATUR para la resolución del Timetabling Problem.

4.5 Traducción del grafo en un timetable

Luego de colorear, el último paso es transformar el grafo G coloreado a un *timetable* que llamaremos T(G) (o T solo).

Un grafo de conflictos σ-coloreado es equivalente a un horario con σ *timeslots*. Si X(G) = k, entonces podemos transformar G a un horario con k *timeslots*.

En la Ilustración 3 vemos el resultado del grafo 6-coloreado, y tenemos un total de 6 *timeslots* .

La transformación de un grafo coloreado a un Timetable se realiza en forma directa, ya que en la Sección 4.3 establecimos que un vértice corresponde a una Clase de nuestra Escuela, y que el color asignado corresponde a un bloque de la semana, entonces tomando esta información recorremos el grafo coloreado y armamos el Timetable:

Como al vértice 1 se le asignó el color r_1, significa que a la clase 1 (que es “mat1-le1-K1 / p1 “) le corresponde el bloque r_1 (que es “dia 1- hora 1”); luego al

vértice 2 se le asignó el color r_2, por lo tanto a la clase 2 (que es “mat1-le2-K1 / p1 “) le corresponde el bloque r_1 (que es “dia 1- hora 2”). Continuamos traduciendo cada vértice y su color a la clase y bloque correspondiente.

Para el ejemplo de la Sección 4.3 no quedaron clases sin bloque asignado y se obtuvo un número cromático de seis.

	Dia1	Dia2	Dia3
Hora1	1.mat1-le1-K1 / p1	3.mat2-le1-K1 / p2	5.mat3-le1-K1 / p1
Hora2	2.mat1-le2-K1 / p1	4.mat2-le2-K1 / p2	6.mat3-le2-K1 / p1

Curso K1

	Dia1	Dia2	Dia3
Hora1	7.mat1-le1-K2 / p2	9.mat2-le1-K2 / p3	11.mat3-le1-K3 / p4
Hora2	8.mat1-le2-K2 / p2	10.mat2-le2-K2 / p3	12.mat3-le2-K3 / p4

Curso K2

Capítulo 5 – Timetabling utilizando Algoritmos Genéticos

5.1 Introducción

En este Capítulo describimos una representación de un *Timetable* y luego la implementación de un algoritmo genético llamado FET realizado por [Lalescu, 2003] y que utilizaremos en el Capítulo 7 de este trabajo para nuestro caso de estudio.

Elegimos FET porque es un proyecto de código abierto que posibilita la parametrización tanto de la información de input en formato XML, como de los parámetros del algoritmo evolutivo, como la probabilidad de *crossover*, la probabilidad de mutación, y más parámetros de ejecución.

También es una implementación dedicada al Timetabling Problem, que se ajusta muy bien para la aplicación de nuestro caso de estudio.

5.2 Representación del problema

Aquí veremos como representar un *timetable* utilizando la representación propuesta por Lalescu [Lalescu, 2003] de forma que sirva de entrada al algoritmo FET descripto en la siguiente sección.

En esta representación tenemos que cada **cromosoma** representa un *timetable*; y se conforma de un conjunto de genes (la unidad más chica que acarrea información de un cromosoma).

Dentro del cromosoma, hay un **gen** por cada clase en el *timetable*. Este gen representa el bloque asignado de tiempo de la correspondiente clase. Por lo tanto, un cromosoma es un vector de genes, cada uno de ellos representando la hora y el día de comienzo de una clase.

A cada gen se le va a asignar un valor, que se denomina **alelo**, que puede ser un 0 que significa NO-Asignado o un valor entre 1 y n, siendo n la cantidad de *timeslots*.

Primero vamos a establecer algunas clases a modo de ejemplo:

1.mat1-le1-K1 / p1
2.mat1-le2-K1 / p1
3.mat2-le1-K1 / p2
4.mat2-le2-K1 / p2
5.mat3-le1-K1 / p1
6.mat3-le2-K1 / p1
7.mat1-le1-K2 / p2
8.mat1-le2-K2 / p2
9-mat2-le1-K2 / p3
10.mat2-le2-K2 / p3
11.mat3-le1-K2 / p4
12.mat3-le2-K2 / p4

("K" significa Curso, "mat" es de Materia, "p" es de Profesor, "le" es de Lección)

Vemos que tomamos 2 Cursos: K1 y K2 y las materias se dictan 2 horas semanales.

Tenemos la distribución semanal:

	Dia1	***Dia2***	***Dia3***
Hora1			
Hora2			

Un cromosoma está conformado por tantos genes como clases hay para representar. Para nuestro ejemplo tenemos que un cromosoma consta de 12 genes, cada uno representa la clase correspondiente a la lista anterior:

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***

El valor de cada gen está en el rango de 0 .. 6, donde :

0 = NO ASIGNADO
1 = dia1-hora1
2 = dia1-hora2
3 = dia2-hora1
4 = dia2-hora2
5 = dia3-hora1
6 = dia3-hora2

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
1	4	2	0	0	0	3	5	0	1	6	5

ejemplo de cromosoma

Lo que en lenguaje natural significa que a la clase 1 (mat1-le1-K1 / p1) le asigno el bloque 1 (dia1-hora1); a la clase 2 le asigno el bloque 4, a la clase 3 el bloque 2, a la clase 4 NO asignada; y así siguiendo hasta la última clase.

Este cromosoma es una posible solución para el ejemplo dado.

5.3 Algoritmo genético aplicado al timetabling

Pretendemos crear una población de individuos, donde cada uno representa un posible *timetable*. Comenzando con una población inicial, se simula la evolución natural mediante mejores candidatos para la solución óptima. La principal presunción es que, como los mejores individuos son los más probables de ser elegidos para la reproducción y son los que sobrevivirán por períodos más largos, los mejores individuos se desarrollan y se hacen fuertes por generaciones. Entonces, esto resulta en la posibilidad de obtener la mejor solución al problema de *timetabling*, representado por el mejor individuo (o con más '*fitness*'). [Lee, 2000]

El algoritmo genético FET utiliza el método de evolución basado en *three tournament selection* (selección de torneo de 3):

> Selecciona un cromosoma candidato de la siguiente manera: a partir de la población actual, se eligen aleatoriamente tres cromosomas. Si se diera el caso que sólo se necesita una simple propagación (es decir, la conservación de un cromosoma) o la mutación de un cromosoma, el mejor cromosoma de los tres, es decir el que tenga mejor "*fitness*", se elegiría para esta tarea. Si la operación considerada fuera de *crossover*, elegimos los mejores dos cromosomas de los tres.

El pseudo-código de FET es:

1)**[start]** Genera una población aleatoria de n cromosomas (posibles soluciones al problema)
2)**[fitness]** Evaluación del *fitness* f(x) de cada cromosoma x en la población
3)**[new population]** Creación de nueva población repitiendo los siguientes pasos, hasta que se completa la población nueva:

a **a) [selection]** Selecciona dos cromosomas padre de la población, de acuerdo su *fitness* (a mayor *fitness*, mayor chance de ser elegido)
b) [crossover] Con una *crossover probability* (probabilidad de cruza), se cruzan los padres para formar un nuevo *offspring* (hijo). Si no se realiza la cruza, el *offspring* es una copia exacta o clon de los padres.
c) [mutation] Con una *mutation probability* (probabilidad de mutación), alternar posiciones en el cromosoma hijo.

4)**[replace]** Usar la población generada para una nueva ejecución del algoritmo
5)**[test]** Si se satisface la condición de terminación, **stop** (terminó) y retorno la población actual como mejor solución.
6)**[loop]** Volver al paso 2)

Describiremos cada una de las etapas del algoritmo utilizando el ejemplo dado en la Sección 5.2.

1) [start] Inicialización: RANDOM, significa que cada cada cromosoma se genera con valores aleatorios.

Aquí se define también el valor de los parámetros tamaño de la población y número de generaciones para que termine el algoritmo, por ej:

Población: 10 cromosomas

Generaciones: 40 . Cantidad de repeticiones del algoritmo genético.

2) [fitness]

La función de *fitness* se computa simplemente contando el número de conflictos con restricciones *hard* , lo cual se hace sumando las Unallocated constraints del cromosoma C con las teacher-exhaustions y las student-exhaustions del mismo cromosoma:

fitness (C)= *weight* * (*unallocated*(C) + *teacher-exhaustions*(C) +

student-exhaustions(C))

donde *weight* (peso) es un número entre 0 y 1, generalmente 1.0

- ***Unallocated constraints***. Aquí damos una valoración numérica como penalización por las Clases que no fueron asignadas a un bloque. Asignamos a cada gen (clase) no asignado, es decir que tiene el alelo 0, a una hora el valor 10000 (un valor muy alto).

Definimos una función unallocated (g), que da para el Gen G ,

$$\text{unallocated (G)} = \begin{cases} 10000, & \text{si el Gen g tiene el valor 0} \\ 0, & \text{caso contrario} \end{cases}$$

Luego definimos la función unallocated (C), que da para el Cromosoma C,

unallocated (C) = SUMA (unallocated(x)) , 1<= g <= 12

- **Teacher Exhaustions.** Usamos la función teacher-exhaustions, que calcula el número de veces que los profesores tienen que dictar lecciones al mismo tiempo.

Esta función recibe el Cromosoma C como parámetro y por cada gen G realiza la suma de bloques que se solapan para ese profesor.

- **Student Exhaustions.** Usamos la función student-exhaustions, que calcula el número de clases que están asignadas al mismo dia y misma hora para el mismo curso.

3) [new population]

3a) [selection]

El pseudo-código de la selección por 3-torneo es:

```
selección_3torneo (poblacion_actual, cromo1, cromo2, cromo3)

k = rand()%100;
si ( k < MUTATION_PROBABILITY ) entonces

        mutación(cromo1)

sino
        si (k < CROSSOVER_PROBABILITY + MUTATION_PROBABILITY) entonces

                cruza(poblacion_actual, cromo1, cromo2)

        sino
                propagación(poblacion_actual, cromo1)
```

siendo CROSSOVER_PROBABILITY y MUTATION_PROBABILITY parámetros que recibe el algoritmo FET en forma de valores de prioridad para el procedimiento de *crossover* o de mutación.

Se necesita una función de comparación para definir cuándo un cromosoma es mejor que otro, [Lalescu, 2003]le llama *better*(C1, C2).

$$better(C1, C2) = \begin{cases} \text{True}, & \text{si } fitness(C1) > fitness(C2) \\ \text{False}, & \text{caso contrario} \end{cases}$$

Esta función retorna verdadero si el *fitness* del cromosoma C1 es mejor que el *fitness* del cromosoma C2, falso en caso contrario.

3b) [crossover]

El operador de cruza en un punto (*crossover operator*)[Coello-Coello, 2006] espera como argumentos dos cromosomas y crea uno nuevo, combinando la parte de más a la izquierda del primer argumento con la parte más derecha del segundo argumento. Los cromosomas descendientes combinarán las cualidades de sus padres en un mejor cromosoma. Estos cromosomas están sujetos a subsecuentes *crossovers* y mutaciones. La función de selección juega un rol importante aquí, promoviendo a los mejores cromosomas. El comportamiento del operador de

crossover se ve a continuación:

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
1	4	2	0	1	0	3	5	0	1	6	5

cromosoma C1

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
0	3	6	0	4	4	0	1	0	2	2	3

cromosoma C2

Punto de cruza

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
1	4	2	0	4	4	0	1	0	2	2	3

cromosoma Hijo de C1 y C2

3b) [mutation]

La mutación (*Mutation*) [Coello-Coello, 2006] es la alteración de un gen de un cromosoma.

La mutación que utiliza FET es: Intercambiar dos genes del cromosoma en forma aleatoria.

Se supone que la mutación introduce un cambio aleatorio en la población actual. Esto es necesario para prevenir que la población se concentre en una zona pequeña, convergiendo a un óptimo local en vez de converger al óptimo global (por ej la solución óptima).

Ejemplo:

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
0	3	6	0	4	4	0	1	0	2	2	3

cromosoma C

intercambiando los valores del gen 3 con el gen 8 nos queda el cromosoma:

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
0	3	1	0	4	4	0	6	0	2	2	3

cromosoma C mutado

4) [replace]

Ahora podemos reemplazar total o parcialmente la población anterior y estar listos para la próxima iteración del algoritmo.

Utilizamos el término *Elitism* (Elitismo) que determina que el mejor cromosoma de la población anterior se debe mantener como está en la nueva población, mientras que los demás cromosomas se generan mediante *crossover* y *mutation* de los cromosomas viejos.

Por ejemplo tenemos:

Población: 10 cromosomas

Población i : C10, C20, C5, C4, C7, C11, C8, C3, C34, C29

Población i+1 : C10, C3, C4, C12, C23, C9, C6, C33, C5, C15

Notar que en la Población i+1 se quitaron cromosomas (C20, C7, C11, C8, C34 y C29) pero se agregaron otros (C12, C23, C9, C6, C33 y C15), mientras que se reordenaron C3, C4 y C5.

Tomamos la población ordenada de forma que el primer elemento desde la izquierda es el Mejor Cromosoma, y es el que se mantiene para la siguiente población (en el ej. sería C10).

5) [test]

Aquí chequeamos que el número de iteraciones del algoritmo no supere el valor máximo que definimos en la fase de inicialización previa. También definimos un tiempo límite en segundos para la terminación de la ejecución del algoritmo.

Por ejemplo:

Max_Generations: 40 . Cantidad de repeticiones del algoritmo genético.

Timelimit: 180 segundos. Tiempo límite que se permite para encontrar una solución.

5.4 Traducción de una población a varios timetables

Luego de iterar durante 40 generaciones, finaliza el algoritmo de la sección anterior aplicando como entrada el ejemplo de la Sección 5.2, obtenemos una población de 10 cromosomas, o posibles soluciones, (que definimos en la etapa **[start]** del algoritmo); de la cual la primera de ellas es la mejor y tiene como valor esperado de *fitness* igual a cero (0).

Por ejemplo tenemos un cromosoma que da el valor ideal de *fitness* y es :

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
1	3	5	2	4	6	1	2	3	4	5	6

cromosoma C con mejor fitness

que traducimos a un *Timetable* utilizando la representación explicada en la Sección 5.2, es decir, tenemos en el Cromosoma C que el gen 1 (“mat1-le1-K1 / p1”) el algoritmo le asignó el valor 1 (“dia1-hora1”); al gen 2 (“mat1-le1-K1 / p1”) le asignó el valor 3 (“dia2-hora1”) y así continuando hasta el gen 12 que es el último. Notar que en este caso no hay valores en 0 (“NO ASIGNADO”), por lo que la solución fué la óptima.

Tenemos como resultado:

	Dia1	***Dia2***	***Dia3***
Hora1	mat1-le1-K1 / p1	mat1-le2-K1 / p1	mat2-le1-K1 / p2
Hora2	mat2-le2-K1 / p2	mat3-le1-K1 / p1	mat3-le2-K1 / p1

curso K1

	Dia1	***Dia2***	***Dia3***
Hora1	mat1-le1-K2 / p2	mat2-le1-K2 / p3	mat3-le1-K2 / p4
Hora2	mat1-le2-K2 / p2	mat2-le2-K2 / p3	mat3-le2-K2 / p4

•curso K2

Este proceso lo automatizamos en el Capítulo 7 en la Fase 4 del Plan de pruebas donde leemos el archivo resultado del algoritmo FET y lo traducimos de esta manera a nuestro Timetable.

Capítulo 6 – Timetabling utilizando Satisfactibilidad

6.1 Introducción

En este capítulo haremos una representación de un *timetable* como una Fórmula Normal Conjuntiva FNC utilizando el método propuesto por [Kenneth-Chin Fat, 2004]. Para el mejor entendimiento de este método aplicaremos un caso sencillo de ejemplo que en la sección 6.4 resolvemos en forma manual. Luego en la sección 6.5 mostramos algoritmos que permiten obtener un Modelo para esta fórmula utilizando la computadora.

En la Lógica Proposicional no hay muchas maneras de decir las cosas, éstas suceden o no suceden. Para poder modelar este conocimiento contamos con las variables booleanas que son la única “estructura de datos” .

Para el caso del school timetabling definimos variables booleanas cuyos valores van a ser asignados con 0 ó 1 para obtener un Modelo que luego lo traduciremos a un horario (*timetable*).

6.2 Representación del Problema

La variable booleana $x_{k,i,j,d,t}$ modelizada por [Kenneth-Chin Fat, 2004] indica que para el curso *k* el par materia-profesor *i* está dictando la lección *j* el día *d* en el horario *t*.

Siendo el rango de cada subíndice el definido en el Capítulo 3 de este trabajo.

Por ejemplo, poniendo:

$x_{1,2,3,4,5}$ **declara** que para el curso 1, materia-profesor 2 dicta lección 3 el día

4 en el horario 5.

- $x_{5,4,3,2,1}$ **declara que no sucede** que para el curso 1, materia-profesor 2 dicta lección 3 el día 4 en el horario 5.

Tomaremos como un caso de ejemplo:

K=1 Un sólo curso que lo llamamos “K1”,

U=1 Las materias sólo dictan 1 lección semanal,

D=4 Tenemos 4 pares materia-profesor

Aquí armamos un currículum del curso K1 para las cuatro pares materia-profesor del ejemplo, donde cada una se dicta una hora semanal.

curso\mat-prof	1	2	3	4
k1	1	1	1	1

currículum del ejemplo

Tomando el currículum vemos cómo queda la matriz de alocación para el único curso que hay:

k1	j1
1	1
2	1
3	1
4	1

matriz alocación Ak1 del curso k1

Clases del ejemplo

1. mat1-le1-K1 / p1
2. mat2-le1-K1 / p2
3. mat3-le1-K1 / p1
4. mat4-le1-K1 / p2

(“K” significa Curso, “mat” es de Materia, “p” es de Profesor, “le” es de Lección)

S=2 Utilizamos dos días de clases:

1. día1
2. día2

T=2 y dos horas diarias:

1. hora1.
2. hora2.

Las variables booleanas que utilizamos entonces son:

1. $x_{1,1,1,1,1}$
2. $x_{1,1,1,1,2}$
3. $x_{1,1,1,2,1}$
4. $x_{1,1,1,2,2}$
5. $x_{1,2,1,1,1}$
6. $x_{1,2,1,1,2}$
7. $x_{1,2,1,2,1}$
8. $x_{1,2,1,2,2}$
9. $x_{1,3,1,1,1}$
10. $x_{1,3,1,1,2}$
11. $x_{1,3,1,2,1}$
12. $x_{1,3,1,2,2}$
13. $x_{1,4,1,1,1}$
14. $x_{1,4,1,1,2}$
15. $x_{1,4,1,2,1}$
16. $x_{1,4,1,2,2}$

La primer variable declara que en el curso 1 (“K1”) se dicta el par materia-profesor 1 (“mat1-le1-K1 / p1”) siendo la lección la número 1 en el día 1 (“dia1”) y en la primer hora (“hora1”), luego la segunda variable indica la combinación del curso 1 con la materia-profesor 1, la leccion 1 y el dia 1 pero con la segunda hora.

6.3 Escritura de un timetable como una FNC

En esta Sección representaremos las posibles soluciones o *Timetables* a las 4 Clases ejemplo de la sección anterior, para esto escribiremos cada *timetable* como una FNC (Fórmula Normal Conjuntiva) como hacen en el trabajo de [Kenneth-Chin Fat, 2004].

Primero aplicamos las relaciones SameTeacher y TeacherAvailable a nuestro ejemplo, quedando:

1)SameTeacher(1,1) =1
2)SameTeacher(1,2) =0
3)SameTeacher(1,3) =1
4)SameTeacher(1,4) =0
5)SameTeacher(2,1) =0
6)SameTeacher(2,2) =1
7)SameTeacher(2,3) =0
8)SameTeacher(2,4) =1
9)SameTeacher(3,1) =1
10)SameTeacher(3,2) =0
11)SameTeacher(3,3) =1
12)SameTeacher(3,4) =0
13)SameTeacher(4,1) =0
14)SameTeacher(4,2) =1
15)SameTeacher(4,3) =0
16)SameTeacher(4,4) =1

Donde dice SameTeacher(1,3)=1 se lee: “el par materia-profesor 1 (“mat1-le1-K1/p1”) y el par 3 (“mat3-le1-K1/p1”) comparten el mismo profesor “p1”.”

Donde dice SameTeacher(4,3)=0 significa que el par materia-profesor 4 (“mat4-le1-K1 / p2”) y el par 3 (“mat3-le1-K1/p1”) son dictadas por diferentes profesores, “p2” y “p1” .

Ahora vemos la asignación de la relación TeacherAvailable, suponiendo que todos los profesores están disponibles para cualquier día y hora de nuestro ejemplo:

- TeacherAvailable(1,1,1) = 1
- TeacherAvailable(1,1,2) = 1
- TeacherAvailable(1,2,1) = 1
- TeacherAvailable(1,2,2) = 1
- TeacherAvailable(2,1,1) = 1
- TeacherAvailable(2,1,2) = 1
- TeacherAvailable(2,2,1) = 1
- TeacherAvailable(2,2,2) = 1
- TeacherAvailable(3,1,1) = 1
- TeacherAvailable(3,1,2) = 1
- TeacherAvailable(3,2,1) = 1
- TeacherAvailable(3,2,2) = 1
- TeacherAvailable(4,1,1) = 1
- TeacherAvailable(4,1,2) = 1
- TeacherAvailable(4,2,1) = 1
- TeacherAvailable(4,2,2) = 1

Interpretando la relación TeacherAvailable(1,1,1)=1 en lenguaje natural significa que el par materia-profesor 1 (“mat1-le1-K1 / p1”) está disponible para dictar en el día 1 (“dia1”) y en la hora 1 (“hora1”).

Recordando la definición de esta relación dada en el Capítulo 3 Sección 3.4, el primer argumento es el par materia-profesor, el segundo es el día y el tercero es la hora.

Definición de la FNC

Modelado de las restricciones como cláusulas

Pasamos ahora a definir las restricciones (*constraints*) que ocurren en el problema de *Timetabling* (definidas en el Capítulo 2) y las modelaremos como cláusulas Ci que formarán parte de la fórmula FNC que servirá de entrada a los métodos de resolución aplicados en las secciones 6.3 y 6.4.

Completeness constraints: cada lección de una materia es obligatoria. Cada combinación de curso k, materia-profesor i y número de lección j que corresponde a $A_k(i,j)=1$ debería ser dictada en un período que la materia-profesor esté disponible.

$$\bigwedge_{k} \bigwedge_{i} \bigwedge_{j \mid A_k(i,j)=1} \bigvee_{d} \bigvee_{t \mid TeacherAvailable(i,d,t)=1} x_{k,i,j,d,t}$$

Donde aparece $j \mid A_k(i,j)=1$ se lee como todas las lecciones j que se dictan al curso k en las materias-profesor i . Esta información se toma de la matriz de alocación Ak.

El subíndice $t \mid TeacherAvailable(i,d,t)=1$ representa al profesor t que está disponible para el dictado de la materia i el dia y hora dados por d y t.

i=1 $(x_{1,1,1,1,1} \vee x_{1,1,1,1,2} \vee x_{1,1,1,2,1} \vee x_{1,1,1,2,2}) \wedge$ cláusula 1.-

i=2 $(x_{1,2,1,1,1} \vee x_{1,2,1,1,2} \vee x_{1,2,1,2,1} \vee x_{1,2,1,2,2}) \wedge$ cláusula 2.-

i=3 $(x_{1,3,1,1,1} \vee x_{1,3,1,1,2} \vee x_{1,2,1,2,1} \vee x_{1,3,1,2,2}) \wedge$ cláusula 3.-

i=4 $(x_{1,4,1,1,1} \vee x_{1,4,1,1,2} \vee x_{1,2,1,2,1} \vee x_{1,4,1,2,2}) \wedge$ cláusula 4.-

Tenemos un total de 4 restricciones.

Lectura intuitiva:

La cláusula 1 establece que la materia-profesor 1

se dicta al curso 1 el día 1 en la primer hora ó

se dicta al curso 1 el día 1 en la segunda hora ó

se dicta al curso 1 el día 2 en la primer hora ó

se dicta al curso 1 el día 2 en la segunda hora

y

La cláusula 2 establece que la materia-profesor 2

se dicta al curso 1 el día 1 en la primer hora ó

se dicta al curso 1 el día 1 en la segunda hora ó

se dicta al curso 1 el día 2 en la primer hora ó

se dicta al curso 1 el día 2 en la segunda hora

y

Lo mismo ocurre con la cláusula 3 para la materia-profesor 3

y la cláusula 4 para la materia-profesor 4.

Uniqueness constraints: las interpretamos como: “cada lección se asigna una sola vez”. Si una lección en particular pudiera se asignada a diferentes horarios, solamente permitimos ser asignada a uno solo de esos horarios.

Si la lección $x_{k,i,j,d,t}$ ocurre (o sea $x_{k,i,j,d,t}$ es V), entonces la misma lección no puede ocurrir en otro período.

$$\bigwedge_{k} \bigwedge_{i} \bigwedge_{j \mid A_k(i,j)=1} \bigwedge_{d,d'} \bigwedge_{\substack{t \mid \text{TeacherAvailable}(i,d,t)=1 \\ t' \mid \text{TeacherAvailable}(i,d,t')=1}} \neg x_{k,i,j,d,t} \vee \neg x_{k,i,j,d',t'}$$

materia i=1 dias d=d'=1

t=1, t'=1 $(\neg x_{1,1,1,1,1} \vee \neg x_{1,1,1,1,1}) \wedge$ cláusula 5.-

t=1, t'=2 $(\neg x_{1,1,1,1,1} \vee \neg x_{1,1,1,1,2}) \wedge$ cláusula 6.-

t=2, t'=1 $(\neg x_{1,1,1,1,2} \vee \neg x_{1,1,1,1,1}) \wedge$ cláusula 7.-

t=2, t'=2 $(\neg x_{1,1,1,1,2} \vee \neg x_{1,1,1,1,2}) \wedge$ cláusula 8.-

materia i=1 dias d=1,d'=2

t=1, t'=1 $(\neg x_{1,1,1,1,1} \vee \neg x_{1,1,1,2,1}) \wedge$ cláusula 9.-

t=1, t'=2 $(\neg x_{1,1,1,1,1} \vee \neg x_{1,1,1,2,2}) \wedge$ cláusula 10.-

t=2, t'=1 $(\neg x_{1,1,1,1,2} \vee \neg x_{1,1,1,2,1}) \wedge$ cláusula 11.-

t=2, t'=2 $(\neg x_{1,1,1,2,2} \vee \neg x_{1,1,1,2,2}) \wedge$ cláusula 12.-

De igual manera seguimos generando cláusulas cuando d=2, d'=1 y d=2, d'=2 para la materia 1.

Se realiza de igual forma para las materias 2, 3 y 4.

Tenemos entonces un total de 64 restricciones para nuestro ejemplo.

Lectura intuitiva:

La cláusula 9 establece que la materia 1 no se dicta al curso 1 el día 1 en la primer hora ó que no se dicta al curso 1 el día 2 en la primer hora

y

La cláusula 10 establece que la materia 1 no se dicta al curso 1 el día 1 en la primer hora ó que no se dicta al curso 1 el día 2 en la segunda hora

y

La cláusula 11 establece que la materia 1 no se dicta al curso 1 el día 1 en la segunda hora ó que no se dicta al curso 1 el día 2 en la primer hora

y

La cláusula 12 establece que la materia 1 no se dicta al curso 1 el día 2 en la segunda hora ó que no se dicta al curso 1 el día 2 en la segunda hora.

De la misma maner se leen las demás cláusulas que representan Uniqueness Constraints.

Teacher Constraints. Se entienden como: “cada profesor puede tener a lo sumo una lección a la vez”.

Planteamos las teacher constraints de la siguiente forma:

$$\bigwedge_{k,k'} \bigwedge_{i,i' \mid SameTeacher(i,i')=1} \bigwedge_{\substack{j \mid A_k(i,j)=1 \\ j' \mid A_{k'}(i',j')=1}} \bigwedge_{d} \bigwedge_{t \mid TeacherAvailable(i,d,t)=1} \neg x_{k,i,j,d,t} \vee \neg x_{k',i',j',d,t}$$

En términos técnicos, comparamos todos los pares materia-profesor i' y i del mismo profesor.

Para plantear estas restricciones, tenemos que por cada par de cursos k y k', y por cada par de número de lecciones dictadas por esos materia-profesores, y por cada período i y i' disponibles en la semana, ambas lecciones no pueden ocurrir en el mismo período.

Ya que la materia i y i' son dictadas por el mismo profesor , o sea SameTeacher(i,i')=1, estas restricciones nos aseguran que cada materia-profesor puede tener a lo sumo una lección en el mismo horario.

$$i=1,\ i'=3\ \ (\neg x_{1,1,1,1,1} \vee \neg x_{1,3,1,1,1}) \wedge$$

$$(\neg x_{1,1,1,1,2} \vee \neg x_{1,3,1,1,2}) \wedge$$

$$(\neg x_{1,1,1,2,1} \vee \neg x_{1,3,1,2,1}) \wedge$$

$(\neg x_{1,1,1,2,2} \vee \neg x_{1,3,1,2,2}) \wedge$

Igual para las materias i, i':

i=2, i'=4

i=3, i'=1

i=4, i'=2

Tenemos un total de 16 restricciones de este tipo.

Lectura intuitiva:

La primer cláusula define que la materia 1 del curso 1 no puede dictarse el día 1 en la primer hora ó que la materia 3 del curso 1 no puede dictarse el día 1 en la primer hora porque son dictadas por el mismo profesor

y

La segunda cláusula define que la materia 1 del curso 1 no puede dictarse el día 1 en la segunda hora ó que la materia 3 del curso 1 no puede dictarse el día 1 en la segunda hora porque son dictadas por el mismo profesor

y

así sucesivamente para las demás pares de materias que dicta el mismo profesor.

Student constraints: cada alumno puede tener a lo sumo una lección en el mismo horario.

Tomamos los pares materia-profesor de cada curso. Si ambos enseñan en este curso y no refieren al mismo profesor, entonces para todo par de lecciones dadas por estos materia-profesor a este curso, solamente uno puede ocurrir en los periodos que ambos están disponibles. En los horarios que sólo uno de estos materia-profesor está disponible, no puede ocurrir que ambos dictan en aquel curso.

$$\bigwedge_{k} \bigwedge_{i,i' \mid SameTeacher(i,i')=0} \bigwedge_{\substack{j \mid A_k(i,j)=1 \\ j' \mid A_k(i',j')=1}} \bigwedge_{d} \bigwedge_{t \mid TeacherAvailable(i,d,t)=1 \atop TeacherAvailable(i',d,t)=1} \neg x_{k,i,j,d,t} \vee \neg x_{k,i',j',d,t}$$

$$\begin{aligned} i=1, i'=2 \quad & (\neg x_{1,1,1,1,1} \vee \neg x_{1,2,1,1,1}) \wedge \\ & (\neg x_{1,1,1,1,2} \vee \neg x_{1,2,1,1,2}) \wedge \\ & (\neg x_{1,1,1,2,1} \vee \neg x_{1,2,1,2,1}) \wedge \\ & (\neg x_{1,1,1,2,2} \vee \neg x_{1,2,1,2,2}) \wedge \end{aligned}$$

Igual para :

i=1, i'=4

i=2, i'=1

i=2, i'=3

i=3, i'=2

i=3, i'=4

i=4, i'=1

i=4, i'=3

Para los cuales se cumple que el profesor de la materia i es distinto al que dicta en la materia i', o sea la relación SameTeacher(i,i') es igual a 0.

Tenemos un total de 32 restricciones del tipo Student Constraints.

La FNC que obtuvimos para las 4 Clases del ejemplo para asignar a 4 bloques semanales, está compuesta por un total de 116 cláusulas que representan las combinaciones posibles para obtener un horario correcto. Las cláusulas fueron clasificadas:

4 cláusulas de Completeness Constraints.

64 cláusulas de Uniqueness Constraints.

16 clásulas de Teacher Constraints y

32 cláusulas de Student Constraints.

6.4 Testeo de Satisfactibilidad

En [Hamilton, 1981] se describe la utilización de las tablas de verdad para hallar si una fórmula proposicional es Verdadera o Falsa, basado en la noción de satisfactibilidad de fórmulas atómicas y la funció de verdad de los conectivos.

Para el armado de la tabla colocamos en las primeras 16 columnas cada una de las variables que utilizamos, y luego las cláusulas que conforman la FNC, que para nuestro ejemplo alcanzan las 116 cláusulas.

El número de renglones de la tabla de verdad es de 2^n, con n número de variables proposicionales. Para el ejemplo anterior tenemos un total de K x D x U x S x T = 16 variables, y la tabla tendrá 65536 combinaciones posibles de valores de verdad.

Colocamos un identificador a cada una de las cláusulas de la FNC, que van del 1. al 116.

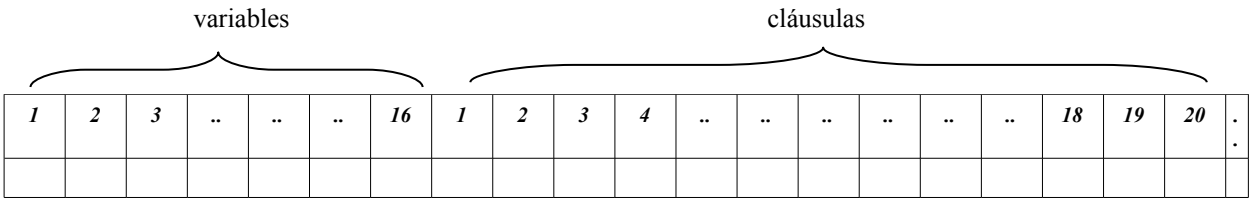

Tabla de Verdad ejemplo.

Ahora buscamos una asignación a cada cláusula en un intento por que el renglón de Verdadero. Si conseguimos la asignación, la FNC es satisfactible y dicha asignación es la solución al problema de Timetabling.

Comenzamos con las **Completeness constraints** que son cláusulas con una sola variable $x_{k,i,j,d,t}$ que debe tener el valor V (Verdadero), en el ejemplo son las cláusulas de la 1. a la 4. Resulta natural empezar por acá.

Luego seguimos con las cláusulas con una sola variable $\neg x_{k,i,j,d,t}$ asignando el

valor F (Falso).

1	*2*	*3*	*..*	*..*	*..*	*16*	*1*	*2*	*3*	*4*	*..*	*..*	*..*	*..*	*..*	*..*	*18*	*19*	*20*	
V	F	F	F	..	..	..							..	..	..	F	V	V	V	**V**

Una vez que completamos la fila de la tabla vemos si el resultado de la operación ∧ aplicada a toda la fila da como resultado Verdadero tenemos un resultado.

Vemos que un posible resultado entonces es:

variable 1 = V

variable 7 = V

variable 12 = V

variable 14 = V

y las demás en F.

El timetable resultante es:

	Dia1	***Dia2***
Hora1	1.mat1-le1-K1 / p1	7.mat2-le1-K1 / p2
Hora2	14.mat4-le1-K1 / p2	12.mat3-le1-K1 / p1

Curso K1

6.5 Resolvedores SAT

Una vez que planteamos todas las restricciones anteriores, se puede generar un horario mediante un *Solver* asignando variables.

Los *Solvers* son programas que implementan algoritmos que resuelven SAT. Estos algoritmos se dividen en:

- Algoritmos SAT Incompletos, basados en búsqueda local. Son capaces de

probar satisfactibilidad en muchas fórmulas generadas aleatoriamente.

- Algoritmos Completos de *Backtrack Search* (búsqueda “hacia atrás”) , se basan en el algoritmo DPLL (Davis-Putnam-Loveland-Longmann) [DPLL,1962]. Han sido utilizados para resolver un gran número de instancias SAT del mundo real.

El método de Davis-Putnam es uno de los métodos más prácticos para el problema de *Satisfiability* (SAT) de la lógica proposicional. Está basado en Resolución [Robinson, 1960].

Brevemente, el método trabaja de la siguiente manera:

Mantiene una asignación parcial ∞ e intenta extenderla asignando valores a variables no asignadas. El valor puede ser impuesto por el valor de las otras variables bajo ∞ o elegido.

Hace así hasta que ∞ es una asignación completa en tal caso la fórmula es satisfacible o que exista una cláusula que da el valor F bajo ∞.

En este último caso el algoritmo hace backtracking, es decir “des-asigna” las últimas variables asignadas hasta que encuentra una variable que puede ser asignada a otro valor. Si tal variable no existe la fórmula de entrada al algoritmo es insatisfacible, sino continúa con el proceso de backtracking.

Método Davis-Putnam [DPLL,1962]

Sea S una fórmula FNC y ∞ una asignación vacía (o parcial) . El algoritmo DPLL de abajo retorna V (Verdadero) y ∞ en cuyo caso S es satisfacible y ∞ es un Modelo, ó F (Falso) y ∞ = {} en cuyo caso S es insatisfacible.

El método consiste en:

2 condiciones: chequear las dos condiciones siguientes:

1. Si S está vacia, retornar V y ∞
2. Si S contiene la cláusula vacía, retornar F y {}

2 reglas: aplicar una de las dos reglas siguientes (elegir una cualquiera, si se puede aplicar más de una):

1) Regla de cláusula Unitaria: Si S contiene una cláusula Unitaria {L} (cláusula que contiene un sólo literal) entonces borrar de S cada cláusula que contiene a L y borrar ¬L de cada cláusula de S donde ésta aparezca
2) Regla de Splitting: generación de subproblemas donde se le asigna a L valores alternativos V o F y se llama al algoritmo DPLL agregando como cláusula {L} o {¬L} a la fórmula S.

Veamos un ejemplo de la aplicación del algoritmo DPLL aplicado a una FNC (que en este caso no representa un *timetable*).

Dado la fórmula FNC S, tal que

S = { {a, b, d } cláusula 1
{ ¬a, b, e} cláusula 2
{ ¬b } cláusula 3
}

Tomamos L = {¬b} la única cláusula unitaria que hay, aplicando la primer regla (regla cláusula unitaria) incluímos L en el resultado ∞ y S nos queda :

S = { {a, d } cláusula 1
{ ¬a, e} cláusula 2
}

∞ = { {¬b} }

Ahora aplicamos la regla 2 (de splitting) tomando como L el literal a y volvemos a aplicar el algoritmo desde el principio pero ahora usando como S :

S = { {a, d } cláusula 1

{ ¬a, e} cláusula 2

{ a } cláusula 4

}

Aplico la primer regla usando L = { a }

S = {

{ e } cláusula 2

}

∞ = { {¬b} , { a } }

Finalmente aplico la primer regla, dando:

S = { }

∞ = { {¬b} , { a }, { e } }

Como S está vacía obtengo el resultado V, por lo que la fórmula S es satisfacible y la asignación resultante es:

b = F

a = V

e = V

SATO

Para el caso de estudio de este trabajo se utilizó una implementación muy eficiente del método Davis-Putnam llamado SATO (Satisfiability Testing Optimized)[Zhang].

Elegimos esta implementación por ser recomendado por las buenas mediciones dadas en la Organización DIMACS [DIMACS, 1992].

En el trabajo de [Zhang, 1999] hacen uso de la estructura trie (o

discrimination trees) para representar términos de primer orden.

SATO es un *solver* Completo: puede mostrar que una instancia no tiene solución.

Estructura Trie Data

La estructura trie fué utilizada por primera vez en [Kleer, 1992] para representar conjuntos de cláusulas proposicionales.

Se asume que cada variable tiene un índice único, que es entero positivo. El índice de una variable negada es la negación del índice de ésa variable. Una cláusula se representa como una lista de índices de los literales de la cláusula.

La estructura trie data es simple. Es un árbol cuyas enganches marcan los índices de los literales y cuyas hojas tienen una marca de terminación. Una cláusula se representa como el camino desde la raiz hasta una hoja de forma que cada enganche es el indice del literal de la cláusula.

Si dos cláusulas (representadas como listas de enteros ordenadas en forma ascendente por su valor absoluto) tienen el mismo prefijo de longitud n, entonces comparten un camino de longitud n en la estrucutra trie.

La representación trie de las cláusulas proposicionales tiene varias ventajas para el método Davis-Putnam:

- Las cláusulas duplicadas son eliminadas automáticamente cuando se construye la estructura trie.
- Se reduce el uso de la memoria por los prefijos de cláusulas compartidas.
- Las cláusulas unitarias se encuentran rápidamente.
- La operación de unit propagation se puede realizar de forma eficiente.
- Algunas cláusulas subsumidas pueden ser eliminadas al momento de construir el árbol.

A continuación vemos el seudocódigo del algoritmo SATO implementado en forma recursiva.

Algoritmo Recursivo

```
function Satisfacible ( conjunto cláusulas S ) return boolean

   /* Propagación Unitaria */

   repeat
        for each clausula unitaria L en S do

                delete from S cada cláusula que contiene a L
                delete ¬L de cada cláusula de S donde ésta aparezca

        od

        if S está vacia then
                return TRUE

        else if la cláusual vacía está en S then

                return FALSE

        fi

    until no haya más cambios por realizar

   /*  splitting */

   elegir un literal L que ocurra en S
   if Satisfacible ( S  U   {L} ) then
        return TRUE

   else if Satisfacible ( S  U  {¬L} ) then

        return TRUE

   else
        return FALSE
   fi

end function
```

Capítulo 7 – Resultados experimentales

7.1 Introducción

En este capítulo mostramos los resultados obtenidos en la ejecución de las técnicas de Inteligencia Artificial estudiadas aplicadas al caso de estudio. La información comparativa la mostramos en forma de tablas para la mejor comprensión de las similitudes y diferencias entre las tecnicas.

Se estudiaron los requerimientos de la escuela y se utilizaron datos reales del ciclo lectivo del año anterior (2006).

Como paso previo a las pruebas, conseguimos implementaciones de código abierto que compilamos e instalamos en una PC convencional. Tales implementaciones corresponden al algoritmo de coloración del grafo Dsatur [Brelaz, 1979], algoritmos genéticos [Lalescu, 2003] y un resolvedor de SAT llamado SATO presentado en [SAT2004].

Para Recocido Simulado [Abramson, 1991] y Búsqueda Tabú [Schaerf, 1995] nombradas en el Capítulo 2 hay implementaciones realizadas para la Competencia de Timetabling explicada en el Apéndice A, pero resuelven el problema de University Timetabling que no se ajustaban exactamente a nuestro caso.

Generamos archivos para el caso de estudio, que sirven de entrada a cada una de las implementaciones. Tenemos en cuenta para las comparaciones el tiempo de respuesta de cada uno y la calidad del Timetable (para los casos que se encontró) comparada a la solución hecha manualmente por el encargado de la escuela.

Para armar los correspondientes archivos, desarrollamos una aplicación general que permite de manera gráfica y fácil ingresar el tamaño del problema, las restricciones y elegir el algoritmo que vamos a probar.

7.2 Requerimientos del caso de estudio

En el Capítulo 3 de este trabajo modelamos las variables que utiliza la persona encargada de resolver el problema de Timetabling de la Escuela Secundaria de nuestro estudio.

A partir de esa información tuvimos que tomar decisiones sobre cuales de ésas variables, por ejemplo la cantidad de clases que se dictan, eran importantes o no y qué restricciones obligatorias (o duras) y opcionales valoraríamos ya que son muchas y algunas muy complejas.

Para lograr una comparación efectiva y útil de las técnicas presentadas anteriormente decidimos que los algoritmos probados en lo que resta de éste Capítulo debían cumplir con las restricciones obligatorias de un problema de *timetabling*, ellas son:

- No asignar dos clases al mismo *timeslot*
- Un profesor no puede dictar en dos cursos al mismo tiempo.

También utilizamos las restricciones opcionales mínimas de modo de lograr la obtención de un Timetable que pueda ser medido en Calidad y en Tiempo, comparado luego con el hecho en forma manual; tales restricciones son:

a) Preferencias de horarios por parte de los profesores.

b) Posibilidad o no de dejar *timeslots* sin asignación de materia, es decir, tener o no horas libres.

Según la restricción a) dividimos el plan de pruebas en 2 variantes: Sin preferencias y Con preferencias.

Caso Sin preferencias significa que realizamos las pruebas teniendo en cuenta sólo las restricciones duras, es decir , que no haya ningún tipo de solapamiento de horarios de los profesores o de las lecciones a dictarse en los cursos.

Caso Con preferencias agrega al caso anterior una restricción liviana que se refiere a las preferencias de los profesores por determinados horarios para el dictado de clases. Tal información se determinó sobre la base de la planilla real de los horarios asignados por la escuela a los profesores.

A continuación describimos en forma de tabla los requerimientos que quedaron establecidos para el caso de estudio:

Nro	Requerimiento	Caso de estudio
1	Tamaño del problema	143 clases, 25 timeslots, 6 cursos, 19 profesores y 8 materias por curso.
2	Duración de la clase	1 Timeslot
3	Definición de timeslot	Cada slot representa un período del día (hay 5 slots por día)
4	Restricciones Duras	Las 2 restricciones nombradas en esta sección
5	Restricciones Opcionales	Las restricciones a) y b) antes mencionadas

1. Tamaño del problema. Se refiere al número total de clases a las que se les va a asignar bloques. El tamaño define de algún modo la complejidad del problema.
2. Duración de la clase. Resolución mínima de tiempo, es decir cuán largo es un período – *timeslot*, generalmente entre 1 y 2 horas, la duración definida para cada *timeslot* en el problema de *timetabling*.
3. Definición de *timeslot*. El período de asignación y la resolución mínima determina el número de timeslots requeridos.

7.3 Representación de los datos

En esta Sección mostramos en forma de tablas cómo representamos la información de la Escuela Secundaria para aplicar los algoritmos explicados en los Capítulos 4, 5 y 6. También mostramos qué información manejará y cómo la almacenará nuestra aplicación. Por último definimos una función de puntuación que nos será muy útil para poder comparar de acuerdo al valor resultado los distintos *timetables* obtenidos.

En el siguiente cuadro describimos cómo representamos la información de la escuela en cada uno de los algoritmos probados.

	Coloración Grafo	SAT	AG
Clases	vértices	variable	gen
Restricciones	arista	cláusula	función fitness
Timeslot	color [0..MaxTimeSlots]	subindices dia y hora	[Unallocated, 0 .. MaxTimeSlots]
Timetable	grafo	FNC	cromosoma
Timetabling Problem	asignar colores al grafo	asignar true / false a la FNC	asignar valores a los genes del cromosoma

Tabla 7.3.1

Luego describimos las estructuras de datos y el tamaño que ocupan en memoria dentro de nuestra aplicación para poder almacenar y procesar la información de la Escuela.

- La X indica que se carga esa estructura para el algoritmo que se va a testear, una O indica que no se utiliza.
- Entre paréntesis se indica la longitud (un número entero) ocupada por la estructura de datos.

Nombre	***Grafo***	***Satisfacibilidad***	***Genéticos***
Materias [1]	X (143)	X (143)	X (19)
Bloques semanales [1]	X (25)	X (25)	X (25)
Curriculum	O	X (6x19)	O
Matriz SameTeacher [2]	X (47x47)	X (47x47)	O
Matriz TeacherAvailable[2]	X (47x47)	X (47x47)	O
Matriz Alocación [2]	O	X (114x4)	O
Colores	X (25)	O	O
Matriz Aristas	X (47x47)	O	O
Nodos	X (143)	O	O
Variables [2]	O	X (11400)	O
Cláusulas [2]	O	X (198618)	O
Actividades	O	O	X (143)

Tabla 7.3.2

1 La información de materias o clases y bloques es obligatoria para todos los algoritmos.

2 Esta información la definimos en el Capítulo 6 de Satisfactibilidad.

Estructura de datos Timetable

En nuestra aplicación definimos la estructura de datos Timetable[curso,dia,hora] en la que se va a guardar en forma de matriz con tres indices un registro que denominamos *Timeslot* donde quedará la asignación de un bloque de tiempo i a una Clase j de la Escuela.

Con esta estructura unificamos la representación de un *Timetable*, sin importar qué técnica o algoritmo de timetabling hayamos aplicado para obtenerlo. Además, teniendo esta única estructura nos simplificó la definición de una función de puntuación que mida la calidad del *Timetable*.

En notación Pascal, la definición es la siguiente:

```
Timeslot = record
      clase: integer;    //1..CANT_MAT, 0 = VACIO
      bloque : integer;   //1..CANT_HRS, 0 = VACIO
      end;

Timetable = array[1..MAX_CURSOS,1..MAX_DIAS,1..MAX_HRS] of ^Timeslot;
```

Siendo las Constantes MAX_CURSOS, MAX_DIAS y MAX_HRS las definidas en el Capítulo 3, Sección 3.3.

Para más legibilidad utilizaremos la notación (id_clase, id_bloque) para representar un Timeslot.

Función de Puntuación

Para medir la calidad del Timetable utilizamos una función objetivo presentada por [Swee-Chuan,2003] que se basa en puntos de penalización.

En el trabajo de Swee-Chuan presentan un Framework Orientado a Objetos para la utilización genérica de algoritmos en el School Timetabling Problem, y utilizan dicha función como una forma automática de calcular el grado de *optimization problem*, que definimos en el Capítulo 1, Sección 1.2.

Una solución es óptima cuando la suma de las penalizaciones converge a 0.

Acción	Valor (puntos)
– Clase asignada en un timeslot NO preferido	1
– Clase NO asignada (hueco en el *timetable*)	5
– Materia con más de 2 hrs en un dia y un curso	10

Los valores de puntuación elegidos son una medida arbitraria que intenta penalizar con más peso las restricciones livianas consideradas más importantes por el encargado de realizar el Timetable en la Escuela Secundaria; por esta razón es que se asigna poca penalización (1) a una clase que se asigna en un Timeslot que el profesor no informó en sus Preferencias Horarias; y se asigna un total de 10 puntos a una Materia de un curso que se asignó en más de 2 hrs en el mismo dia, esto es en general lo que sucede en las Escuelas Secundarias que no se dicta más de esa cantidad de horas una Materia para el mismo curso y mismo Día.

Mostramos a continuación la función que implementamos para realizar el cálculo:

```
function puntuar_resultado: integer;
{ Implementacion de la funcion de puntuacion del Capitulo 7 }
var pts_hueco,pts_mat2hr,pts_preferencia:integer;
begin

    pts_hueco:=0;
    pts_mat2hr:=0;
    pts_preferencia:=0;

    // cargo el timetable resultado
      leer_archivo_resultado(arch_dim);

    // evaluo cantidad de huecos por dia
    // 1 hueco = 5 puntos
    pts_hueco := calcular_puntos_xhueco;

    // evaluo cantidad de materias que sobrepasan las 2 hrs diarias
    // 1 materia = 10 puntos
    pts_mat2hr := calcular_puntos_2hrs;

    // evaluo preferencias cumplidas.
    // tiene preferencia = 0, no lo eligio = 1 punto.
    pts_preferencia:=calcular_puntos_preferencia;

    puntuar_resultado:= pts_hueco +pts_mat2hr + pts_preferencia;

end;
```

7.4 Plan de Pruebas

En las secciones previas definimos los requerimientos de la Escuela Secundaria de estudio. Luego dimos la representación de la información requerida por los algoritmos a comparar así como la representación en forma de estructuras de datos que maneja la aplicación que desarrollamos.

A continuación describimos el plan de pruebas que hemos diseñado y llevado a cabo. Aquí nos encontramos con varias complicaciones, la primera era que debiamos compilar los códigos fuente de los programas correspondientes a las diferentes técnicas.

Para realizar las pruebas se utilizó un equipo PC con procesador AMD Athlon Xp 2400, 512 MB de memoria RAM, sistema operativo Kubuntu Linux 7.04.

Se utiliza el sistema operativo Linux porque es el ambiente natural de las aplicaciones probadas, las cuales tienen disponible el código fuente que compilan perfectamente con las herramientas provistas por el sistema. Esto no quita que puedan ejecutarse en otras plataformas, aunque no se garantiza la mejor performance de las aplicaciones.

Con el ambiente de ejecución definido, elegimos el lenguaje de programación Pascal para nuestra aplicación pues funciona sobre este sistema

operativo, es fácil de usar y realiza un manejo muy eficiente de archivos (o sea lectura-escritura en disco rígido). Como ambiente de desarrollo o IDE (Interface Development) utilizamos uno llamado Lazarus con el cual podemos realizar aplicaciones de escritorio o de línea de comandos con un menú.

Otro problema importante que debimos sortear es la diversidad de formatos de archivos que manejan tanto de entrada como de salida a los programas que compilamos y ejecutamos, que nos llevó por ejemplo a utilizar librerías auxiliares para el manejo de XML en nuestro IDE.

También tuvimos que interiorizarnos en la lógica de programación de los programas utilizados ya que contaban con poca o nula documentación, tal es el caso de FET, en el cual debimos leer el código fuente hasta encontrar el funcionamiento de la función de aptitud o *fitness* que evalúa con cada cromosoma.

Para esto último debimos leer código fuente en los lenguajes de programación C y C++ que no manejábamos.

Después de varios cambios en la cantidad y el alcance, finalmente dividimos el Plan de Pruebas en 5 Fases o etapas bien definidas:

- **Fase 1: Representar los datos de entrada.**
- **Fase 2: Generar los archivos de entrada para los diferentes algoritmos.**
- **Fase 3: Ejecutar los programas DSATUR, SATO y FET.**
- **Fase 4: Ingresar los resultados para imprimir el timetable de forma legible por el usuario.**
- **Fase 5: A los timetables obtenidos aplicar la función de puntuación para medir su calidad.**

De esta forma, logramos aplicar el Plan a cada uno de los algoritmos estudiados utilizando nuestra aplicación para la interrelación y para el manejo de lo que es el Timetable solución.

A continuación pasamos a describir cada una de estas Fases para cada uno de los algoritmos.

Fase 1: Representar los datos de entrada

El primer paso es volcar la información de la escuela real (presentada en el Capítulo 3 de este trabajo) en archivos de texto cuyo formato definimos de la siguiente manera:

materias.txt

Contiene tres columnas separadas por un espacio entre columnas:

id-materia id-curso cant-hrs

por ejemplo

1 1 4

significa que la materia con Id 1, dicta en el curso con Id 1 cuatro horas semanales.

materias_con_nombre.txt

Describe cada una de las materias que se dictan, utilizando 2 filas del archivo por materia, siendo la primer fila para una materia:

id-materia nombre

y la fila continua inferior las preferencias horarias del profesor que dicta dicha materia; en esta fila se guardan tantas columnas como timeslots haya en el Timetable separados por una “,” (coma) y en cada una de las posiciones un 0 indica que la materia no se puede asignar en ese timeslot y un 1 que sí se puede.

por ejemplo

1 Soc-Terzaghi

1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

describe con la primer fila una materia con Id 1, llamada “Soc-Terzaghi”, y con la

segunda fila los 25 timeslots para nuestra escuela, indicando que el profesor Terzaghi puede dictar en cualquiera de ellos.

horarios.txt

Aquí indicamos los timeslots que contiene el Timetable que queremos resolver. Este archivo tiene 3 columnas:

id-timeslot	dia	hora

por ejemplo

20	4	5

es decir, el timeslot 20 corresponde a la quinta hora del dia 4.

Ventajas:

Estos archivos pueden ser modificados con cualquier editor de textos para agregar, modificar o quitar información de los profesores, de las clases o de las restricciones involucradas.

La información de estos archivos las guardamos en estructuras de datos que definimos:

```
materia_nombre = record
        id:integer;
        nombre:string;    //descripcion
        preferencias: string; //cadena de 0s y 1s
        profesor:string;      // nombre del profesor
        materia:string;       // nombre de la materia
        end;

materia = record
        nombre: integer;  //id de la materia puede por ej ser de 1..19 //( NO es el id de la
                                clase)
        // para saber el verdadero nombre de la materia o preferencias del profesor
        // consultamos el tipo materia_nombre
        profesor: integer;  //id del profesor
        curso: integer;       //id del curso
        cant_hrs: integer;  // total de hrs que se dicta semanalmente
        asignada: boolean;  // campo auxiliar para busqueda en Alg. Gen.
        end;

timeslot = record
        nombre: integer;   //id del timeslot ej 1..25
        dia: integer;
        hora:integer;
        end;
```

Fase 2: Generar los archivos de entrada para los diferentes algoritmos.

Para esta Fase utilizamos nuestra aplicación que permite leer los archivos materias.txt, materias_con_nombre.txt y horarios.txt definidos en la Fase 1, y carga en memoria las diferentes estructuras de vectores y matrices que mostramos en la Sección 7.3.

Entrada a nuestra aplicación:

Información de la escuela en los archivos materias.txt, materias_con_nombre.txt y horarios.txt.

Salida de nuestra aplicación:

	Dsatur	SATO	FET
formato de archivo	DIMACS	DIMACS	XML
parámetros	cant vertices, cant aristas	cant variables, cant cláusulas	tamaño cromosoma, tamaño población
contenido	Aristas del grafo	Cláusulas de la FNC	Lecciones a asignar
nombre del archivo	prueba.col	prueba.fnc	ejemplo_real.fet

La aplicación que desarrollamos permite la utilización de grafos y de cláusulas proposicionales. El manejo de los archivos de algoritmos genéticos esta incorporado en el programa FET, es decir, que en la tabla anterior la 3er columna en realidad es la salida de la aplicación FET, pero la incluimos en la tabla para mejor visualización.

Los formatos de archivo utilizados siguen los estándares internacionales que se pueden consultar en [DIM93] ,[DIMACS] y [W3C, 2002].

Para el manejo de Grafos implementamos las estructuras de datos nodo y grafo de la siguiente forma:

```
nodo = record
    materia: integer;    //1..CANT_MAT, 0 = VACIO
    timeslot : integer;   //1..CANT_HRS, 0 = VACIO
    end;

grafo = array[1..MAX_NODOS] of nodo;

matriz_adyacentes = array[1..MAX_NODOS,1..MAX_NODOS] of integer;
// 1 significa nodo i es adyacente del nodo j, 0 cc.
```

Nota 1: Para generar la matriz de adyacentes, sólo tomamos “aristas de ida” es decir cuando j>i y colocamos un 1 si las materias i y j son del mismo curso o del mismo profesor, sino colocamos 0.

Esto es una indicación dada por el estándar DIMACS para el archivo. En nuestro caso tenemos un maximo de 47x47=2209 aristas, de las cuales nos quedan 2097 si seguimos con la indicación del estándar.

```
procedure inicializar_adjacentes;
var i,j:integer;
begin

  for i:= 1 to MAX_NODOS do
    for j:= 1 to MAX_NODOS do
       mat_aristas[i,j]:= 0;

  for i:= 1 to CANT_MAT do begin
    for j:= 1 to CANT_MAT do begin

      // tomo solo aristas "de ida", indicacion del standard
      if (j>= i) then begin

        //la diagonal no es adyacente
        if (i=j) then mat_aristas[i,j]:= 0
        else begin
          //agrego arcos si son materias del mismo curso o
          //materias del mismo profesor.
            if (mat_IC[i,j] = 1) or (mat_IP[i,j]=1)then
              mat_aristas[i,j] := 1
            else mat_aristas[i,j] := 0;
          end;

       end;
    end;
  end;

end;
```

Con el grafo armado y la matriz de adyacentes inicializada, utilizamos el procedimiento que arma el archivo que le llamamos grafo_a_archivo_dim :

```
procedure grafo_a_archivo_dim;
var tot_aristas:integer;
i,j:integer;
begin
assign(arch_dim,nom_arch_destino);
rewrite(arch_dim);

tot_aristas:= calcular_tot_aristas;

//Encabezado archivo DIMACS con el total de aristas
writeln(arch_dim,'p edge ',CANT_MAT,' ',tot_aristas);

for i:= 1 to CANT_MAT do
  for j:= 1 to CANT_MAT do

    if mat_aristas[i,j]=1 then
       // según el estandar cada linea comienza con la 'e'
       writeln(arch_dim, 'e ', i, ' ', j);


close(arch_dim);

end;
```

Por cuestión de espacio mostramos aquí sólo las primeras líneas del archivo resultado del procedimiento anterior:

```
p edge 143 2097
e 1 2
e 1 3
e 1 4
e 1 5
e 1 6
e 1 7
e 1 8
e 1 9
e 1 10
e 1 11
e 1 12
e 1 13
e 1 14
e 1 15
```

prueba.col

Donde se indica en la primer línea el total de nodos del grafo 143 y el total de aristas del mismo que asciende a 2097. Luego cada línea debe comenzar con la letra e del inglés *edge* (arista) que representa una arista del grafo, por ejemplo

```
e 1 2
```

indica que hay una arista del nodo 1 al nodo 2.

Para el manejo de Satisfacibilidad declaramos las siguientes estructuras de datos que representan una variable y una cláusula proposicional:

```
//variable proposicional
variable = record

        cur : integer;
        mat : integer;
        lec : integer;
        dia : integer;
        hor : integer;
        signo : integer; // 0=positivo, 1=negativo
        indice : integer; //resultado de la reduccion a un indice

        end;

//clausula proposicional
lista_var=^nodo_var;
nodo_var = record
    dato : variable;
    sig : lista_var;
    end;

//FNC
lista_cla=^nodo_cla;
nodo_cla = record
    clausula : lista_var;
    sig : lista_cla;
    end;
```

Necesitamos también representar las restricciones explicadas en la Sección 6.3 dentro de nuestra aplicación, para eso implementamos por ejemplo el procedimiento que arma las cláusulas correspondientes a las TeacherConstraints:

```
// TEACHER CONSTRAINTS
procedure llenar_teacher(var lista_teacher:lista_cla);
var k,i,j:integer;
k1,i1,j1:integer;
l_aux : lista_var;
v_izq : variable;
v_der : variable;
ok:boolean;
begin

inicializar_cla(lista_teacher);

for k:= 1 to KA do begin        //cursos
for k1:= 1 to KA do begin        //cursos

    for i:= 1 to DE do begin            //materias
    for i1:= 1 to DE do begin            //materias

     if ( mat_ST[i,i1] = 1) then begin

        for j:= 1 to U do begin       //lecciones
        for j1:= 1 to U do begin       //lecciones

          if ((mat_aloc[k,i,j].dicta = 1) and (mat_aloc[k1,i1,j1].dicta = 1)) then begin

           //inicializar_var(l_aux);
            ok:=false;
            if (k<>k1)  then ok:=true
            else if (i<>i1) then ok:=true
                 else if (j<>j1) then ok:=true
                      else ok:=false;

            if ok then begin

             for dia_act:= 1 to S do BEGIN//dias
              for hora_act:=1 to T do BEGIN //horas

                 if (mat_TA[i,dia_act,hora_act]=1) then begin

                   //writeln('Inserto');
                   inicializar_var(l_aux);

                    v_izq.cur := k;
                    v_izq.mat := i;
                    v_izq.lec := j;
                    v_izq.dia := dia_act;
                    v_izq.hor := hora_act;
                    v_izq.signo := 1;
                    v_izq.indice := 0;

                    v_der.cur := k1;
                    v_der.mat := i1;
                    v_der.lec := j1;
                    v_der.dia := dia_act;
                    v_der.hor := hora_act;
                    v_der.signo := 1;
                    v_der.indice := 0;

                    insertar_var(l_aux,v_izq);
                    insertar_var(l_aux,v_der);

                    insertar_cla(lista_teacher,l_aux);

                 end; //if

              END; //for
             END; //for
        end; //if
      end; //if
      end; //for
      end; //for
   end; //if
  end; //for
  end; //for
  end; //for
  end; //for
end;
```

El total de cláusulas se obtiene sumando las TeacherConstraints, StudentConstraints, TeacherAvailabilityConstraints, CompletenessConstraints, UniquenessConstraints y NegationConstraints.

Luego de generar las listas correspondientes a las cláusulas, ya tenemos el total de variables y el total de cláusulas de la FNC armada. Con esta información utilizamos el siguiente procedimiento que llamamos lista_cla_a_archivo:

```
procedure lista_cla_a_archivo(var arch:archivo_dimacs; l:lista_cla);
{ Genera un archivo DIMACS utilizando la FNC que ingresa como parametro}
var v:variable;
aux:lista_cla;
aux_var : lista_var;
txt:string;
txt_aux:string;

begin

  assign(arch,nombre_arch);
  rewrite(arch);

  // Encabezado DIMACS del archivo con los totales
  writeln(ARCH,'p cnf ',tot_var,' ',tot_cla);

  aux:= l;
  while aux <> nil do
    begin
      aux_var:= aux^.clausula;
      txt:='';

        while aux_var <> nil do begin

         v.signo:=aux_var^.dato.signo;
         if v.signo = 1 then
            txt:=txt + '-';

         v.indice:=aux_var^.dato.indice;
         str(v.indice,txt_aux);
         txt:=txt + txt_aux + ' ';

         aux_var:= aux_var^.sig;

         end;
        // según el estandard cada linea finaliza en 0
        txt:= txt + '0';
        writeln(ARCH,txt);

      aux:= aux^.sig;
      end;
      close(arch);

end;
```

El mismo produce como salida el archivo que sigue el estándar DIMACS:

```
p cnf 11400 198618
-1 -26 0
-2 -27 0
-3 -28 0
-4 -29 0
-5 -30 0
-6 -31 0
-7 -32 0
-8 -33 0
-9 -34 0
-10 -35 0
```

```
-11 -36 0
-12 -37 0
-13 -38 0
```

En este caso mostramos una parte del mismo porque es demasiado extenso.

La primer línea indica que el total de variables proposicionales en el mismo es de 11400 y que la cantidad total de cláusulas supera las ciento noventa mil. Luego cada linea indica cada cláusula proposicional que integra la FNC, cada par de números es el resultado de la reducción a un índice que se aplicó a cada variable, como indicamos a continuación.

Reducción a un índice

Los *solvers* no pueden manejar variables como las definidas en el Capítulo 6 directamente. Ellos requieren enteros, excepto cero, para representar variables, por lo que por ejemplo 102 significa que la variable 102 se asigna el valor verdadero. Y poner -102 significa que la variable 102 se setea en falso.

Entonces debemos transformar la variable $x_{k,i,j,d,t}$ en la variable x_n , por lo que si queremos tener $x_{k,i,j,d,t}$ en true ponemos *n* y si queremos representar $-x_{k,i,j,d,t}$ colocamos -*n*.

Vamos a transformar los índices usando la reducción a un índice que propone [Kenneth-Chin Fat, 2004]:

$$n = t + \mathrm{T} \times (d\text{-}1) + \mathrm{T} \times \mathrm{S} \times (j\text{-}1) + \mathrm{T} \times \mathrm{S} \times \mathrm{U} \times (i\text{-}1) + \mathrm{T} \times \mathrm{S} \times \mathrm{U} \times \mathrm{D} \times (k\text{-}1)$$

En esta Fase, aplicamos la reducción a un índice a la lista de 11400 variables que necesitamos para las combinaciones de Clases y Horarios.

Aquí mostramos la función que implementamos para aplicar reducción a un índice:

```
procedure reduccion_indice_var(var l:lista_var);
// Procedimiento que reduce a un indice cada variable de la clausula.

 var aux:lista_var;

   n,t1,d1,i,k,k1:integer;
   v_aux:ptr_var;
 begin
   aux:= l;
   if aux = nil then   mostrar('Lista Vacia.');

   while aux<> nil do
    begin
        t1:= aux^.dato.hor;
        d1:=aux^.dato.dia;
        i:=aux^.dato.lec;
        k:=aux^.dato.mat;
        k1:=aux^.dato.cur;

        n := t1 + T *(d1-1)+ T * S * (i-1) + T * S * U * (k-1)+ T *S*U*DE*(k1-1);

        aux^.dato.indice:=n;
        aux:= aux^.sig;

    end;
 end;
```

Fase 3: Ejecutar los programas DSATUR, SATO y FET.

Primero compilamos e instalamos en la PC los programas correspondientes a la implementación de Dsatur [Culberson], Sato 3.2 [Zhang] y Fet 3.18.2 [Lalescu, 2003] de acuerdo a las instrucciones que venían en los archivos de código fuente.

Ejecutamos en la línea de comandos de Linux,

para Dsatur:

./dsatur prueba.col

Mostrando como salida de consola de texto:

```
J. Culberson's Implementation of
              DSATUR
A program for coloring graphs.
For more information visit the webpages at:

       http://www.cs.ualberta.ca/~joe/Coloring/index.html

This program is available for research and educational purposes only.
There is no warranty of any kind.

       Enjoy!

Do you wish to use the cheat if present? (0-no, 1-yes)
0
```

```
ASCII format
number of vertices = 143
p edge 143 2097
Number of edges = 2097 edges read = 2097
GRAPH SETUP cpu =  0.00
Enter seed for search randomization: 64748
 64748
Process pid = 9257

 DSATUR COLORING
Initial Vertex Ordering:
        1 -- inorder
        2 -- random
        3 -- decreasing degree
        4 -- increasing degree
Using: 2
2
CLRS =24        CLRSUM = 5208
Coloring Verified
Coloring time cpu =  0.00
```

Informa que interpretó el archivo de entrada como un grafo de 143 vértices y de 2097 aristas. Informa el ID del proceso en la PC para poder hacer un seguimiento de los recursos utilizados por la ejecución del mismo.

Finalmente nos da a elegir el Orden en que Dsatur va a colorear el grafo. Estas variantes (*inorder*, *random*, *decreasing degree y increasing degree*) están detalladas en [Brelaz,1979], para nuestra escuela no afectaron el resultado final en términos de tiempo de CPU, que también es informado por la salida de la ejecución de este programa, ni en la Calidad del Timetable obtenido.

Y el archivo generado es el siguiente:

```
CLRS 24 FROM DSATUR cpu =  0.00 pid = 16330
 16  15  22  13   8   7   5   1  17   4  19  20  10  11   3   2   6  12  23   9
 14  18  21   4   1   3   2   9  11  12  10  22  21  24  18   6   7  23  20  19
 15  17  13   8  16   5  14  12  18  17  19   4   6   3   2  16   5  15   8  22
 21  13  24  14   9  11  10   7  23   1  20   9   4   7   8  22  23  24  21   1
  6   3  18  20  16  10  12  11  17  19  13  15  14   2   5  16  18  23  20  24
 10  17   3   7   2  14  11   5   9  12  21   1   6  19  22   8  13   4  15  16
 22  15  24   2   8   4   6  12   9  13  10  21  19  20  18  23   3   5   7   1
 17  11  14
```

prueba.col.res

para Sato:

./sato prueba.fnc

Dando la salida de consola de texto:

```
The job "./sato prueba.fnc" started at 000

Input file "prueba.fnc" is open.
Reading clauses in DIMACS's format.
There are 51586 input clauses (4169 unit, 26443 subsumed, 29312 retained).
The maximal index of propositional variables is 11400.
The mallocated memory is     607.98 Kbytes.
There are 20974 clauses in the trie (143 are non-Horn).

Model #1: (indices of true atoms)

   16  42  68  98  311  327  358  397  504  532  562  594  715  746  909  928  970  1206  1230  1264
   1288  1824  1850  1924  1926  1952  1994  2221  2228  2268  2298  2512  2541  2567  2586  2606  2647  3009
3035  3070
   3115  3132  3158  3179  3405  3714  3738  4024  4044  4073  4078  4101  4142  4154  4182  4308  4345  4372
4385  4611
   4637  4714  4740  4752  5106  5146  5163  5191  5425  5509  5543  5811  5846  5873  5877  6006  6038  6066
6087  6203
   6243  6274  6276  6509  6533  6705  6739  6754  7017  7044  7057  7097  7320  7425  7440  7701  7728  7762
7797  8016
   8046  8067  8093  8123  8150  8164  8177  8313  8349  8504  8530  8560  9006  9032  9059  9086  9219  9320
9333  9725
   9743  9770  9782  9922  9944  9973  9999  10009  10030  10061  10088  10201  10229  10515  10527  10553
10916  10942  10958  10987
   11021  11210  11239

The number of found models is 1.

There are 299 branches (1 succeeded, 0 failed).

The search tree path is:
 ( f-11005 f-3020 f-3010 f-3045 f-5420 f-5419 f-9220 f-2592 f-2591 f-2587 f-2517 f-2516 f-2542 f-1838 f-1839 f-1849
f-1813 f-1814 f-3713 f-4608 f-4612 f-4609 f-6512 f-6508 f-6537 f-3106 f-3104 f-3108 f-3107 f-3131 f-3129 f-3133 f-
3156 f-3154 f-3181 f-1213 f-1214 f-1205 f-1238 f-1239 f-1263 f-701 f-704 f-722 f-721 f-726 f-729 f-747 f-2601 f-
2624 f-2622 f-2626 f-2649 f-8324 f-8301 f-8304 f-10224 f-10204 f-10249 f-7418 f-7414 f-7410 f-7415 f-7443 f-7439
f-7435 f-6703 f-6702 f-6704 f-6714 f-6728 f-6727 f-6729 f-6753 f-6752 f-10503 f-10502 f-10528 f-8048 f-8047 f-
8043 f-8042 f-8041 f-8023 f-8022 f-8018 f-8017 f-8073 f-8072 f-8068 f-8098 f-8097 f-9924 f-9923 f-9919 f-9949 f-
9948 f-9974 f-9720 f-9707 f-9718 f-9732 f-9745 f-9757 f-4003 f-4023 f-4019 f-4028 f-4048 f-4053 f-11209 f-11208 f-
11214 f-11234 f-11233 f-9309 f-9308 f-9334 f-5518 f-9012 f-9011 f-9009 f-9007 f-9037 f-9036 f-9034 f-9062 f-9061
f-9087 f-10912 f-10908 f-10917 f-10937 f-10933 f-10962 f-10013 f-10011 f-10005 f-10038 f-10036 f-10063 f-2227 f-
2226 f-2248 f-2243 f-2246 f-2202 f-2201 f-2223 f-2218 f-2252 f-2251 f-2273 f-2277 f-2276 f-1919 f-1902 f-1901 f-
1944 f-1927 f-1969 f-23 f-18 f-17 f-48 f-43 f-73 f-904 f-902 f-910 f-920 f-903 f-929 f-927 f-935 f-945 f-954 f-952 f-960
f-7702 f-7723 f-7722 f-7712 f-7703 f-7727 f-7748 f-7747 f-7737 f-7752 f-7773 f-7772 f-7777 f-7798 f-5802 f-5823 f-
5821 f-5827 f-5848 f-5852 f-8110 f-8104 f-8102 f-8114 f-8125 f-8135 f-8129 f-8127 f-8139 f-8160 f-8154 f-8152 f-
8185 f-8179 f-8515 f-8510 f-8505 f-8540 f-8535 f-8565 f-4702 f-4715 f-4727 f-5147 f-5132 f-5142 f-5141 f-5131 f-
5138 f-5122 f-5107 f-5117 f-5116 f-5113 f-5172 f-5157 f-5167 f-5166 f-5197 f-5182 f-5192 f-7022 f-7007 f-7019 f-
7047 f-7032 f-7072 f-6001 f-6012 f-6016 f-6013 f-6026 f-6037 f-6041 f-6051 f-6062 f-6076 f-6210 f-6201 f-6224 f-
6218 f-6235 f-6226 f-6249 f-6260 f-6251 f-6285 f-4147 f-4133 f-4132 f-4129 f-4126 f-4122 f-4108 f-4107 f-4104 f-
4172 f-4158 f-4157 f-4197 f-4183 f-347 f-333 f-336 f-322 f-308 f-372 f-4310 f-4322 f-4320 f-4335 f-4347 f-4360 f-
519 f-512 f-507 f-544 f-537 f-569 )

---------- Stats ----------
run time (seconds)              0.08
  build time                  0.08
  search time                 0.00
mallocated (K bytes)          2751.18
---------------------------------------
The job "./sato prueba.fnc" ended at 000
```

En la salida por consola que produce el programa SATO podemos leer en la primer parte que reconoció las 11400 variables y muestra la cantidad de memoria utilizada por la estructura.

Luego en el renglón que dice:

Model #1: (indices of true atoms)

que encontró 1 Modelo, es decir una asignación de valores Verdaderos a las 11400 variables que hacen Verdaderas a las 198618 cláusulas de nuestra FNC (Forma Normal Conjuntiva) definida en el Capítulo 6. Esta asignación la guardamos en un archivo que utilizaremos en la siguiente Fase:

```
  16 42 68 98 311 327 358 397 504 532 562 594 715 746 909 928 970 1206 1230 1264
  1288 1824 1850 1924 1926 1952 1994 2221 2228 2268 2298 2512 2541 2567 2586 2606 2647 3009
3035 3070
  3115 3132 3158 3179 3405 3714 3738 4024 4044 4073 4078 4101 4142 4154 4182 4308 4345 4372
4385 4611
  4637 4714 4740 4752 5106 5146 5163 5191 5425 5509 5543 5811 5846 5873 5877 6006 6038 6066
6087 6203
  6243 6274 6276 6509 6533 6705 6739 6754 7017 7044 7057 7097 7320 7425 7440 7701 7728 7762
7797 8016
  8046 8067 8093 8123 8150 8164 8177 8313 8349 8504 8530 8560 9006 9032 9059 9086 9219 9320
9333 9725
  9743 9770 9782 9922 9944 9973 9999 10009 10030 10061 10088 10201 10229 10515 10527 10553
10916 10942 10958 10987
  11021 11210 11239
```

prueba.res

Luego imprime el camino de búsqueda en el árbol de búsqueda definido por [Zhang, 1999] para su implementación del algoritmo Davis-Putnam [DPLL,1962] detallado en el Capítulo 6.

Finalmente imprime el tiempo total de ejecución del programa y la memoria utilizada en la PC.

Para Fet:

Abrimos primero el programa FET escribiendo *fet* en la consola. Luego entramos en la opción “File –> Open” y elegimos el archivo ejemplo_real.fet (mostramos una parte del mismo):

```
<!DOCTYPE FET><FET version="3.18.2">

<Institution_Name>School Timetabling – Pablo Sánchez</Institution_Name>

<Comments>Default comments</Comments>

<Hours_List>
        <Number>5</Number>
        <Name>Hora1</Name>
        <Name>Hora2</Name>
        <Name>Hora3</Name>
        <Name>Hora4</Name>
        <Name>Hora5</Name>
        <Name>Hora6</Name>
</Hours_List>
```

```
<Days_List>
        <Number>5</Number>
        <Name>Lunes</Name>
        <Name>Martes</Name>
        <Name>Miercoles</Name>
        <Name>Jueves</Name>
```

ejemplo_real.fet

Finalmente procedemos a la ejecución del algoritmo genético por la opción "Timetable –> Allocate Hours".

La ejecución va mostrando en pantalla el estado de la asignación, junto con el tiempo y los conflictos que quedan por resolver.

Al finalizar el programa, se genera el archivo timetable.xml con el Timetable resultado.

```
<?xml version="1.0" ?>
<Students_Timetable>

  <Subgroup name="1">
   <Day name="1">
    <Hour name="1">
     <Week1> <Teacher name="Manuele"></Teacher><Subject name="Naturales"></Subject><Subject_Tag
name=""></Subject_Tag></Week1>
     <Week2></Week2>
    </Hour>
    <Hour name="2">
     <Week1> <Teacher name="Solas"></Teacher><Subject name="Artistica"
```

timetable.xml

En el Apéndice damos detalles del formato XML.

En esta fase tenemos los resultados de la ejecución de los algoritmos en distintos archivos como indica la siguiente tabla.

	Dsatur	***Sato***	***Fet***
Nombre archivo del Timetable.	prueba.col.res	prueba.res	timetable.xml

Fase 4: Ingresar los resultados para imprimir el timetable de forma legible por el usuario.

Para Dsatur:

Recorriendo el archivo prueba.col.res, mostrado el la Fase 3, por líneas de izquierda a derecha completamos el Timetable utilizando los vectores de traducción de vértices a materias y colores a Timeslots que definimos en la Fase 2.

Cada posición de uno de los números del archivo prueba.col.res representa un vértice del grafo (en nuestro caso el id de la Clase) y el número mismo es el color que se le asignó en la coloración.

Vemos que el archivo tiene 143 números que nombran a cada una de las 143 Clases definidas en el Capitulo 3 Seccion 3.2, y que cada número está en el rango del 1 al 25, o sea uno de los 25 Timeslots en los que se divide el Timetable.

Leemos la primer posición del archivo (16) como que a la Clase con Id 1(Sociales-Terzaghi Curso7A) dicta en el Timeslot 16 (Dia Jueves, 1er Hora).

La segunda posición del archivo (15) indica que la Clase con Id 2 (Sociales-Terzaghi Curso7A) dicta en el Timeslot 15 (Dia Miércoles, 5ta Hora).

El procedimiento que implementamos es:

```
procedure leer_resultado_coloracion;
var
aux,cad:string;
i:integer;
pos:integer;
nro,resul:integer;
n:nodo;
hora_asig:integer;
cur,dia,hor:integer;
ptr_bloque:^bloque;

begin

//INICIALIZO EL TIMETABLE_RESULTADO
inicializar_timetable;


  assign(arch_dim,nom_arch_resul);
  reset(arch_dim);


   // la primer linea la salteo porque es un titulo
   readLN(arch_dim,cad);
   writeln(cad);

    pos:=0; //la posicion establece que materia es

     while not eof(arch_dim) do
          begin

            // empiezo en la siguiente linea
            readLN(arch_dim,cad);
            writeln(cad);

            i:= 1;
           // pos:=0;
            while (i < length(cad)) do begin

                aux:='';
                while (i < length(cad)) and (cad[i] <> ' ') do begin

                 aux:= aux + cad[i];
                 i:=i + 1;
                end;

                if cad[i]=' ' then
                    //quito todos los blancos
                    while (i < length(cad)) and (cad[i] =' ') do i:= i+1
                   else aux:= aux + cad[i];
```

```
              writeln(aux);
              val(aux,nro,resul);

            if (nro > 0) then begin

              pos:=pos+1;  //posicion del color
              n:=vec_nodo[pos];
              hora_asig := vec_colores[nro].horario;
              n.horario:= hora_asig;

              get_datos_nodo(n,cur,dia,hor);

              new(ptr_bloque);
              ptr_bloque^.materia:=n.materia ;
              ptr_bloque^.horario :=n.horario ;
              org_resultado[cur,dia,hor]:=ptr_bloque;


             end;      // nro > 0

           end; //while  i < length(cad)

           //readLN(arch_dim,cad);

         end;    //while

  close(arch_dim);

  //imprimo el resultado
imprimir_timetable;

end;
```

Tomando los primeros 23 números del archivo tenemos el Timetable para el Curso 7A:

Timetable resultado para el Curso con Id 1 (“7A”):

Timetable[1, 3, 3] = (1, 16)
Timetable[1, 3, 5] = (2, 15)
Timetable[1, 5, 2] = (3, 22)
Timetable[1, 4, 3] = (4, 13)
Timetable[1, 3, 2] = (5, 8)
Timetable[1, 1, 1] = (6, 7)
Timetable[1, 2, 5] = (7, 5)
Timetable[1, 3, 1] = (8, 1)
Timetable[1, 1, 3] = (9, 17)
Timetable[1, 1, 2] = (10, 4)
Timetable[1, 1, 4] = (11, 19)
Timetable[1, 1, 5] = (12, 20)
Timetable[1, 4, 5] = (13, 10)
Timetable[1, 3, 4] = (14, 11)
Timetable[1, 2, 1] = (15, 3)
Timetable[1, 2, 3] = (16, 2)
Timetable[1, 2, 2] = (17, 6)
Timetable[1, 5, 1] = (18, 12)
Timetable[1, 5, 3] = (19, 23)
Timetable[1, 4, 2] = (20, 9)
Timetable[1, 2, 4] = (21, 14)
Timetable[1, 4, 4] = (22, 18)
Timetable[1, 4, 1] = (23, 21)

que visto en forma más gráfica es:

Curso 7A

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
1	Nat-Manuele	Ing-Clark	Art-Solas	Soc-Terzaghi	Cat-Dambrosio
2	Ing-Clark	Nat-Manuele	Mat-Warning	Len-Lonne	Soc-Terzaghi
3	Ing-Clark	Nat-Manuele	Soc-Terzaghi	Cat-Dambrosio	Mat-Warning
4	Len-Lonne	Mat-Warning	Mat-Warning	Len-Lonne	
5	Nat-Manuele	Art-Solas	Soc-Terzaghi	Len-Lonne	

Para SATO:

Cada posición del archivo prueba.res representa cada variable lógica que el Solver SATO asignó el valor Verdadero (True).

Recorremos este archivo traduciendo cada variable a su correspondiente combinación de Materia y Horario, y con cada traducción completamos el Timetable. Para esto creamos el procedimiento leer_archivo_resultado:

```
procedure leer_archivo_resultado(var arch:archivo_dimacs);
var v: variable;
i:integer;
p_var:^bloque;
aux,cad:string;
resul:integer;
cad1:char;
str1:text;
nro:integer;
begin


    assign(arch,nombre_arch_entrada);

    while not eof(arch) do
         begin

          readLN(arch,cad);
          writeln(cad);

         //si uso Jerusat es asi
         //if cad[1] = 'v' then begin
         // i:=3;

         //sino uso SATO
         i:=1;

          while (i < length(cad)) do begin
               aux:='';
               while (i < length(cad)) and (cad[i] <> ' ') do begin
                //aux := concat(cad,cad1);
                aux:= aux + cad[i];
                i:=i + 1;
               end;
               if cad[i]=' ' then i:= i+1
                  else aux:= aux + cad[i];

               //imprimo el valor leido
               //writeln(aux);
               val(aux,nro,resul);

           if (nro > 0) then
            if (vec_var[nro]<> nil) then begin
              //writeln('encontre',nro);

             v:=vec_var[nro]^;

             //imprimo los datos del vector
             //writeln(v.cur,' ',v.mat,' ',v.lec,' ',v.dia,' ',v.hor);

             new(p_var);
             get_datos_materia(v.mat,v.lec, p_var^.materia);
             get_datos_horario(p_var^.horario,v.dia,v.hor);

             org_resultado[v.cur,v.dia,v.hor]:=p_var;


            end;

          end; //while
        end;    //while

    close(arch);

end;
```

Tenemos entonces el archivo prueba.res con el Modelo encontrado, que para nuestra escuela se compone de la asignación T (True) a las 143 Clases identificadas por las variables proposicionales.

Entonces debemos transformar la variable x_n en la variable $x_{k,i,j,d,t}$, por lo que

si tenemos $n = 12$, identifica a un determinado curso K, materia I lección J y dia d y hora t.

Para ver a qué combinación corresponde creamos un archivo auxiliar en nuestra aplicación, al que llamamos variables.dat y que guarda todas las variables encontradas en la Fase 1 junto con el resultado de la reducción a un índice aplicado a cada una de ellas. De esta manera podemos traducir el resultado arrojado por el Solver SAT a la estructura Timetable.

Para FET:

Abrimos el archivo timetable.xml y traducimos cada sección con la etiqueta <subgroup> que identifica al curso, dentro se encuentran las subsecciones <Day>, <Hour> y <Subject> que identifican el Día, Hora y Materia asignada.

Aquí mostramos el procedimiento para traducir del archivo XML a la estructura Timetable de la aplicación Pascal:

```
procedure ParseXML(knoten:TDomNode; var s,curso,dia,hora,prof,mat:string);
var E:TDOMElement;
    N, sub:TDomNode;
    i, zaehler:integer;
    E2:Tdomelement;
//    curso,dia,hora:string;

begin
 E:=knoten as TDomElement;
 N:=E.FirstChild;

 While (N<>Nil) do
  begin

     addToOutput(N,s,curso,dia,hora,prof,mat);
     //curso:= e.GetAttribute('name') ;

     if N.HasChildNodes then
      begin
       I:= N.childnodes.count;
       sub := N.FirstChild;
       E2:= N as TDomElement;
       //dia := E2.GetAttribute('name') ;
       addToOutput(sub,s,curso,dia,hora,prof,mat);

       if sub.haschildnodes then
         begin
           ParseXML(sub,s,curso,dia,hora,prof,mat);
         end;

        for zaehler := 1 to i-1 do
         begin
          sub := sub.NextSibling;
          addToOutput(sub,s,curso,dia,hora,prof,mat);

          if sub.HasChildNodes then
            begin
             ParseXML(sub,s,curso,dia,hora,prof,mat);
            end;
         end;

    end;

    N:=N.NextSibling;


  end; {while}

end;
```

Timetable resultado para el Curso con Id 1 (“7A”):

Timetable[1, 1, 1] = (4, 1)
Timetable[1, 1, 2] = (8, 2)
Timetable[1, 1, 3] = (19, 3)
Timetable[1, 1, 4] = (19, 4)
Timetable[1, 1, 5] = (13, 5)

En forma más gráfica al siguiente Timetable:

Curso 7A

	Lunes	***Martes***	***Miercoles***	***Jueves***	***Viernes***
1	Naturales-Manuele				
2	Artística-Solas				
3	Catequesis-Dambrosio				
4	Catequesis-Dambrosio				
5	Matemática-Warning				

Para la impresión por pantalla del *timetable* implementamos un procedimiento común a todas las técnicas que es:

```
procedure imprimir_timetable;
var k,h,d:integer;
begin
for k:= 1 to CANT_CURSOS do begin
  writeln('Curso: ',k );

  for h:= 1 to CANT_HRS do begin
    for d:= 1 to CANT_DIAS do begin
      if org_resultado[k,d,h] <> nil then begin

        write(' M: ',vec_materias_nombre[ vec_materias[ org_resultado[k,d,h]^.materia ].
nombre ]^.nombre);

        end
           else write('--vacio--');
      end;
    writeln('');
    end;
  writeln('');
end;

end;
```

A continuación vemos un archivo resultado con la impresión del *timetable* hallado por el algoritmo de coloración del grafo:

```
Curso: 1
 M:  Nat-Manuele    M:  Ing-Clark      M:  Art-Solas      M:  Soc-Terzaghi   M:  Cat-Dambrosio
 M:  Ing-Clark      M:  Nat-Manuele    M:  Mat-Warning    M:  Len-Lonne      M:  Soc-Terzaghi
 M:  Ing-Clark      M:  Nat-Manuele    M:  Soc-Terzaghi   M:  Cat-Dambrosio  M:  Mat-Warning
 M:  Len-Lonne      M:  Mat-Warning    M:  Mat-Warning    M:  Len-Lonne      --vacio--
 M:  Nat-Manuele    M:  Art-Solas      M:  Soc-Terzaghi   M:  Len-Lonne      --vacio--

Curso: 2
 M:  Soc-Terzaghi   M:  Art-Solas      M:  Nat-Manuele    M:  Com-Cuenca     M:  Len-Pintos
 M:  Soc-Terzaghi   M:  Art-Solas      M:  Nat-Manuele    M:  Mat-Warning    M:  Len-Pintos
 M:  Soc-Terzaghi   M:  Mat-Warning    M:  Mat-Warning    M:  Len-Pintos     M:  Ing-Sarsfield
```

```
 M:  Soc-Terzaghi    M:  Nat-Manuele    M:  Cat-Dambrosio  M:  Ing-Sarsfield    M:  Len-Pintos
 M:  Cat-Dambrosio   M:  Nat-Manuele    M:  Mat-Warning    M:  Ing-Sarsfield    --vacio--

Curso: 3
 M:  Cat-Ferreyro    M:  Nat-Manuele    M:  Mat-Bruno      M:  Len-Lonne        M:  Art-Reyes
 M:  Nat-Manuele     M:  Mat-Bruno      M:  Soc-Inchausti  M:  Soc-Inchausti   M:  Art-Reyes
 M:  Nat-Manuele     M:  Len-Lonne      M:  Ing-Clark      M:  Soc-Inchausti    M:  Com-Aguerre
 M:  Nat-Manuele     M:  Mat-Bruno      M:  Ing-Clark      M:  Soc-Inchausti    M:  Ing-Clark
 M:  Len-Lonne       M:  Mat-Bruno      M:  Len-Lonne      M:  Cat-Ferreyro     --vacio--

Curso: 4
 M:  Len-Lonne       M:  Len-Lonne      M:  Ing-Sisto      M:  Art-Reyes        M:  Nat-Manuele
 M:  Cat-Ferreyro    M:  Soc-Lorenzo    M:  Ing-Sisto      M:  Mat-Bruno        M:  Nat-Manuele
 M:  Len-Lonne       M:  Soc-Lorenzo    M:  Mat-Bruno      M:  Len-Lonne        M:  Nat-Manuele
 M:  Soc-Lorenzo     M:  Soc-Lorenzo    M:  Com-Aguerre    M:  Mat-Bruno        M:  Nat-Manuele
 M:  Cat-Ferreyro    M:  Ing-Sisto      M:  Mat-Bruno      M:  Art-Reyes        --vacio--

Curso: 5
 M:  Ing-Clark       M:  Mat-Valori     M:  Len-Lonne      M:  Soc-Lorenzo      M:  Ing-Clark
 M:  Len-Lonne       M:  Len-Lonne      M:  Ing-Clark      M:  Nat-Diaz         M:  Mat-Valori
 M:  Nat-Diaz        M:  Mat-Valori     M:  Com-Aguerre    M:  Soc-Lorenzo      M:  Soc-Lorenzo
 M:  Cat-Ferreyro    M:  Art-Solas      M:  Len-Lonne      M:  Mat-Valori       M:  Nat-Diaz
 M:  Art-Solas       M:  Nat-Diaz       M:  Cat-Ferreyro   M:  Soc-Lorenzo      --vacio--

Curso: 6
 M:  Mat-Valori      M:  Nat-Diaz       M:  Cat-Ferreyro   M:  Soc-Inchausti    M:  Art-Solas
 M:  Nat-Diaz        M:  Mat-Valori     M:  Len-Lonne      M:  Com-Cuenca       M:  Soc-Inchausti
 M:  Mat-Valori      M:  Nat-Diaz       M:  Len-Lonne      M:  Ing-Sisto        M:  Ing-Sisto
 M:  Nat-Diaz        M:  Len-Lonne      M:  Cat-Ferreyro   M:  Art-Solas        M:  Soc-Inchausti
 M:  Mat-Valori      M:  Len-Lonne      M:  Soc-Inchausti  M:  Ing-Sisto        --vacio--

                                        resultado_col.txt
```

Cada una de las columnas indica los días de la semana de izquierda a derecha: lunes, martes, miércoles,jueves y viernes.

Cada una de las filas por curso indica el bloque o *timeslot* al que fué asignada cada una de las materias para el curso.

Indicamos los bloques libres con el texto "--vacio--".

Fase 5: A los timetables obtenidos aplicar la función de puntuación para medir su calidad.

Una vez que tenemos armada la estructura de datos Timetable obtenido para cada una de las técnicas aplicadas, pasamos a obtener una medida de la Calidad de los resultados obtenidos para compararlos con la Resolución Manual del Timetabling Problem presentado en el Capítulo 3.

Aquí vemos por ejemplo el Timetable resultado (utilizando Grafos sin preferencias horarias) para el Curso 8B y el correspondiente puntaje asignado.

Curso 8B

	Lunes	***Martes***	***Miércoles***	***Jueves***	***Viernes***
1	Len-Lonne	Len-Lonne	Ing-Sisto	Art-Reyes	Nat-Manuele
2	Cat-Ferreyro	Soc-Lorenzo	Ing-Sisto	Mat-Bruno	Nat-Manuele
3	Len-Lonne	Soc-Lorenzo	Mat-Bruno	Len-Lonne	Nat-Manuele
4	Soc-Lorenzo	Soc-Lorenzo	Com-Aguerre	Mat-Bruno	Nat-Manuele
5	Cat-Ferreyro	Ing-Sisto	Mat-Bruno	Art-Reyes	

Indica la materia que se dicta más de 2 hrs en el día.

Indica un bloque sin asignar del timetable.

Hacemos el cálculo:

1 Timeslot sin asignar el Viernes en la 5 Hora --> 5 puntos de penalización.

2 Materias con más de 2 hrs asignadas para el mismo día, Sociales (Soc-Lorenzo) y Naturales (Nat-Manuele) --> 20 puntos de penalización.

Para este caso tenemos una penalización del Curso de 25 puntos. De la misma forma aplicamos la función a los demás cursos del Timetable, dándonos la penalización total del Timetable resuelto con cada una de las técnicas estudiadas.

7.3 Resultados Obtenidos

La siguiente tabla resume los resultados que obtuvimos:

Area testeada	*Descripción*	*Grafo*	*SAT*	*AG*
Representatividad del problema	Indica el nivel de descripción de la información por parte del algoritmo.	Dificil	Dificil	Fácil
Técnica independiente del problema de timetabling	Indica si el algoritmo se puede utilizar para otro tipo de problema.	Si	Si	No
Cantidad de soluciones por ejecución	Si nos permite elegir entre varios timetables o no.	0-1	0-1	0-N , N=tamaño de la población
Eficiencia en tiempo	Tiempo transcurrido desde el inicio hasta obtener una respuesta.	milisegundos	milisegundos	segundos
Calidad de la solución	Tomamos los valores de la función objetivo a maximizar.	Mala	Buena	Muy Buena

Resultados Experimentales

Los parámetros de interés incluyen tiempo de ejecución del programa y calidad de la solución del mismo.

Aquí no tenemos en cuenta el tiempo requerido para producir los archivos de entrada a los algoritmos y tampoco lo relacionado con la impresión de los resultados tanto en pantalla como en forma de archivos.

PERFORMANCE

El tiempo de ejecución es un parámetro importante en el estudio de problemas NP-completos.

Presentamos aquí los resultados empíricos que hemos obtenido con nuestras pruebas comparando la performance de los 3 algoritmos testeados.

Tiempo de Ejecución (segundos)

Caso	DSATUR	SATO	FET
Sin preferencias	0.5 segs	0.5 segs	0.1 segs
Con preferencias	0.5 segs	0.5 segs	0.1 segs

Los tiempos se determinan tomando como partida la hora de comienzo de ejecución de los algoritmos y fin cuando obtienen una solución o llegan a un timpo máximo de ejecución, que es parámetro de los algoritmos.

CALIDAD

A continuación presentamos los puntajes obtenidos al aplicar la función objetivo sobre los *timetables* obtenidos en la ejecución de cada uno de los programas sobre el caso de estudio. Diferenciamos cuando tenemos o no en cuenta las restricciones de preferencias horarias.

Calidad (función de puntuación sección 7.3)

Caso Sin preferencias

Acción	DSATUR	SATO	FET	Resolución Manual
Lección asignada en un timeslot NO preferido	NO	NO	NO	NO
Lección NO asignada (hueco en el timetable)	35	35	35	35
Materia con más de 2 hrs en un dia y un curso	100	130	40	0
Totales de penalización	135	165	75	35

Caso Con preferencias

Acción	DSATUR	SATO	FET	Resolución Manual
Lección asignada en un timeslot NO preferido	4	0	NO	NO
Lección NO asignada (hueco en el timetable)	35	35	35	35
Materia con más de 2 hrs en un dia y un curso	100	40	40	0
Totales de penalización	139	75	75	35

En ambos casos (con y sin preferencias horarias) para todos los algoritmos la suma de lecciones no asignadas siempre es de 35 puntos. Esto se debe a que los cursos suman cada uno una carga horaria semanal de entre 23 y 24 hrs (siempre queda 1 o 2 *timeslots* libres).

Tenemos también que en la resolución manual realizada por el encargado ya da como resultado un *timetable* correcto que cumple con la tercer restricción de la tabla anterior.

En los tres algoritmos testeados no pudimos llegar a la solución óptima, que para los datos dados por la Escuela Secundaria sería de 35 puntos.

Si quisiéramos un *timetable* completo (en el cual no quedan bloques libres) la solución óptima sería cuando el total de la penalización es 0.

Capítulo 8 – Conclusiones

Tendencias

De acuerdo a los resultados obtenidos podemos confirmar lo que indican las tendencias de la comunidad científica con respecto al uso de Técnicas de Inteligencia Artificial para la resolución de problemas cuyo espacio de búsqueda es grande.

Entre las organizaciones mundiales dedicadas a la resolución de los tipos de problemas NP encontramos a PATAT [PATAT] y a DIMACS [DIMACS] de las cuales damos mas información en el Apéndice A que avalan la utilización de metaheurísticas como los Algoritmos Genéticos.

Aunque hay mucha bibliografía sobre Algoritmos Genéticos ([Coello-Coello,2006]) tuvimos el inconveniente de no encontrar implementaciones en código fuente que fueran específicas para el School Timetabling presentado en esta Tesis.

- No podemos dejar de lado que SAT también tiene puntos fuertes sobre todo por su base lógica, aunque se hace muy difícil (lo mismo que ocurrió en [Kenneth-Chin Fat, 2004]) la representación del problema mediante una sola fórmula proposicional. Se debe hacer uso de “funciones extralógicas” como *Sameteacher* y *TeacherAvailable* de la Sección 3.3. También debemos representar cada posible combinación de timeslot-clase utilizando una variable booleana que representamos en la Sección 6.2. Esto se debe a que el lenguaje de lógica proposicional es de “bajo nivel”: se pueden usar un número limitado de operaciones para escribir todas las restricciones posibles.

La existencia de *solvers* SAT disponibles libremente y evolucionando contínuamente hace atractivo probar utilizando formulaciones SAT para los problemas combinatorios.

- La utilización de la Coloración del Grafo cumplió con el propósito de hallar una asignación básica, pero no permite ampliar a la cantidad de condiciones que debe cumplir un Timetable. Esto se explica con más detalle en la Sección 7.4 de Resultados Obtenidos.

■ Mejoras a los algoritmos.

- Un problema de la utilización de grafos del Capítulo 4, es que no se puede representar las preferencias horarias en el mismo. Siempre tenemos una coloración independiente de las preferencias del profesor. Esto suma muchos puntos de penalización.

No tenemos la posibilidad de pre-asignar colores a los vértices del grafo : Una arista entre dos vértices del grafo hace que no se pueda asignar el mismo *timeslot* a esas dos clases, pero lo que queremos es limitar qué *timeslot* asignar a clases en particular de acuerdo a las preferencias horarias del profesor.

Para lograr una solución a esta cuestión deberíamos modificar el algoritmo Dsatur para que ya no sea un algoritmo general de coloración de grafos sino que sea específico para Timetabling, teniendo en cuenta las preferencias horarias al momento de seleccionar los colores a asignar en el grafo. Ross et al. [Ross, 1997] muestra cómo el algoritmo Brelaz con backtracking en combinación con otras heurísticas puede ser usado para resolver el problema de Examination Timetabling. Para esto agrega las restricciones *near-clash* (en la que dos exámenes con una arista en común se asignan a 2 *timeslots* diferentes) y restricciónes *capacity* (que limitan el número de lugares en cualquier *timeslot*).

– El algoritmo FET del Capítulo 5 cumplió con todas las pruebas realizadas en cuanto a cantidad de soluciones obtenidas, la calidad de las mismas y la representatividad de la información real que maneja el encargado de la escuela del caso de estudio.

  Una posible mejora que buscan los desarrolladores de FET tiene que ver con el tiempo total de ejecución, que lleva un orden de minutos y hasta horas según la carga de restricciones. La causa de esto es que el algoritmo funciona como “fuerza bruta” recorriendo todo el espacio de búsqueda sólo guiándose con las medidas de *fitness* y de los óptimos locales y globales.

  Este parámetro lo buscan mejorar en las competencias como la mostrada en el Apéndice A por la comunidad dedicada a las metaheurísticas de búsqueda.

  Debemos considerar también que este algoritmo calcula varias soluciones

posibles al mismo tiempo.

Hicimos una prueba con datos de ejemplo que vienen con la aplicación, en la cual se definieron 1200 clases, que se dictan en los 5 días de la semana a lo largo de 8 bloques diarios, por 49 profesores. Para esta ejecución el algoritmo tardó 2 horas y 20 minutos. Igual es una muy buena medición comparado con los meses que le lleva a la persona encargada de esa tarea.

- El algoritmo SATO del Capítulo 6 es “independiente del problema”. Ello nos obliga a volcar toda la información de la escuela real en una sola FNC que puede no ser trivial obtener. Una vez obtenida el Modelo puede obtenerse en cuestión de segundos. Esto lo mostramos en detalle en la Sección 7.4 de Resultados Obtenidos.

  Para el caso de Sato se pueden modelar las restricciones livianas de la Sección 1.2 (Timeslot sin asignar y Clases con más de 2 timeslots asignados en el mismo día) como dos *constraints* más que generarían más cláusulas para incorporar a la FNC. Estas cláusulas proposicionales extras se pueden consultar en [Kenneth-Chin Fat, 2004].
- Otra posibilidad que existe para el modelado de las restricciones, en vez del uso de la lógica proposicional, es lo que se llama Cálculo Proposicional [Hamilton,1981]. Este agrega más recursos para representar un problema. Además es la base del paradigma de Programación Lógica que utilizó por ejemplo [Triska, 2008] para implementar con éxito un School Timetabler que denominó SimsTab.

- En cuanto a Sato y a Fet ambos dan buenos resultados que respetan las preferencias horarias en la mayoría de los casos. El problema es que dejan materias con más de 2 hrs en el mismo día, que es algo que la escuela no permite.

■ Cuál es el mejor algoritmo para la resolución del caso de estudio propuesto.

Es difícil elegir un algoritmo. Todos resuelven el School Timetabling.

Tampoco podemos compararlos utilizando alguna medida o estándar internacional porque no existe ninguno en lo que respecta al problema aquí estudiado. Hay aproximaciones realizadas por los organismos PATAT y DIMACS que explicamos en el Apéndice A. Utilizamos como comparación principal los resultados que se obtienen en la resolución manual del timetabling realizado por el encargado del establecimiento educativo.

De las soluciones encontradas, se realizaron con una disponibilidad normal de un profesor. Obtuvimos así resultados coherentes con nuestras expectativas y encontramos una respuesta afirmativa al uso de Técnicas de la Inteligencia Artificial para resolver el School Timetabling.

Si analizamos los resultados obtenidos en la comparación de las técnicas, estos son buenos resultados, pues, son casos de prueba con datos reales.

También concluímos que la factibilidad de obtener buenas soluciones depende en gran medida del algoritmo diseñado y de la herramienta de implementación.

Apéndice A

Competencia Internacional de Timetabling

La Competencia Internacional de Timetabling fué organizada por Metaheuristics Network [Metaheuristics] y tenía como sponsor a la organización PATAT [Patat].

El problema propuesto fué el de crear un Timetable semanal para una Universidad.

Metaheuristics Network es un proyecto de la Comisión Europea, formada por cinco institutos europeos, que buscan comparar y analizar la performance de varias metaheurísticas sobre diferentes problemas de optimización combinatorios, incluyendo el Timetabling.

PATAT (Practice and Theory of Automated Timetabling) es una serie de Series de Conferencias bi-anuales que se mantienen como un Foro para Investigadores y Practicantes del Timetabling para intercambiar ideas.

Detalles de la Competencia

Descripción del Problema

El timetabling problem fué propuesto por Ben Paechter de Metaheuristics Network. Es una reducción del típico problema del University Course Timetabling. Consiste de una serie de eventos a ser asignados en 45 timeslots (5 días de 9 horas cada uno), un conjunto de aulas donde toman lugar los eventos, un conjunto de estudiantes que asisten a los eventos, y un conjunto de características (features) que satisfacen las aulas y que requieren los eventos. Cada estudiante concurre a un número de eventos y cada aula tiene un tamaño o capacidad.

Un Timetable óptimo es aquel en el que todos los eventos han sido asignados a un

timeslot y a un aula por lo que se satisfacen las siguientes Restricciones Duras:

- No hay estudiantes que concurran a más de un evento al mismo tiempo.
- El aula es suficientemente grande para todos los estudiantes que concurren, y satisface las características que necesita el evento
- Sólo hay un evento en cada aula y en cada timeslot

Además, un Timetable candidato se penaliza de igual manera por la ocurrencia de una de las siguientes violaciones de Restricciones Livianas:

- Un estudiante tiene clases en el último bloque del día
- Un estudiante tiene más de 2 clases consecutivas
- Un estudiante tiene una sóla clase en el día.

Valor de las soluciones

Para chequear el valor de una solución:

Es la solución óptima ? Si no entonces la solución no nos sirve.

Si es óptima:

1. Contar el número de ocurrencias de un estudiante que tiene sólo una clase en un día (por ej contar 2 si un estudiante tiene 2 días con sólo una clase)
2. Contar el número de ocurrencias de un estudiante que tiene más de 2 clases consecutivas (3 consecutivas vale 1, 4 consecutivas vale 2, 5 consecutivas vale 3, etc). Clases al final del día que continúan al principio del día siguiente no cuentan como consecutivas.
3. Contar el número de ocurrencias de un estudiante teniendo clases en el último timeslot del día.
4. Sumar los tres contadores para darnos el valor de la solución, cuanto más chico es mejor, el 0 (cero) siempre es posible con las instancias utilizadas en la competencia.

Los organizadores proveen de un programa que chequea la solución para asegurarse que se han entendido las Restricciones.

Instancias del Problema

Las instancias tiene una dificultad variable, pero se espera ser capaz de encontrar una solución óptima para todas ellas dentro del tiempo de computación dado. Todas las instancias tienen al menos una solución perfecta, que es la solución sin violaciones de restricciones, duras o livianas, pero no se espera encontrar dentro del tiempo dado.

Para la evaluación de los algoritmos se dan 20 instancias.

Ganador

Para cada una de las 20 instancias propuestas cada participante presenta la mejor solución encontrada por su algoritmo en el tiempo de CPU especificado. El ganador debe mostrar esos resultados repetidamente en el tiempo de computación dado. Los participantes que no pueden proveer de soluciones óptimas para todas las instancias son excluídos de la competencia. El ganador es el participante que alcanza los mejores resultados en todas las instancias.

Mas precisamente para cada instancia y participante se da un puntaje entre 0 y 1 que se calcula así:

(x – b) / (w-b), siendo x el número de restricciones livianas del participante en esta instancia, b es el número de restricciones livianas violadas del mejor participante en esta instancia, y w es el número de restricciones livianas del peor participante de esta instancia. Estos puntajes se suman para las 20 instancias para cada participante resultando en un puntaje total entre 0 y 20.

El ganador es el participante con menor puntaje.

DIMACS

El centro DIMACS [DIM93] se encarga del estudio teórico y práctico de aplicaciones de Matemáticas Discretas y la ciencia de la computación teórica.

Abarca una amplia variedad de tareas, incluyendo asesorías, desafíos o competencias, y facilitar a los investigadores en distintas áreas, esponsoreando conferencias y workshops.

La investigación fundamental de las Matemáticas Discretas tiene aplicaciones en diversas áreas que incluyen Criptografía, Ingeniería, Redes, y Soporte de Toma de Decisiones.

Los Desafíos de Implementación DIMACS tratan la cuestión de determinar la performance de algoritmos existentes donde el análisis del peor caso es pesimista y en los que los modelos probabilisticos son imposibles: la práctica puede proveer la guía para la performance de los algoritmos donde el análisis teórico falla.

La experimentación trae las preguntas algorítmicas más cerca de los problemas originales que motivaron el trabajo teórico. También se testea las suposiciones hechas sobre los métodos de implementación y las estructuras de datos. Provee una oportunidad de desarrollar y probar instancias de problemas, generadores de instancias y otros métodos de testing y comparación de performance de los algoritmos.

El sitio web incluye información de los Desafíos, como Llamadas para Participar, implementación de algoritmos, bibliografía y más.

Los organizadores también producen series de libros conteniendo los papers seleccionados finalistas de los Desafíos.

En el Segundo Desafío DIMACS del año 1992-1993 se trató los Problemas NP-Duros: Coloración del Grafo y Satisfactibilidad (SAT).

Un propósito del Desafío DIMACS es facilitar el esfuerzo requerido para probar y comparar algoritmos y heurísticas proveyendo una base común de instancias y herramientas de análisis. Para facilitar la tarea, se especificó un formato oficial para el uso en las implementaciones específicas de los algoritmos.

El formato es referido como formato CNF para los archivos de entrada y salida de las aplicaciones.

No era un requerimiento que los participantes usaran dicha especificación, pero las implementaciones compatibles deberían hacer uso de las herramientas de soporte DIMACS.

XML.

El XML, siglas de eXtensible Markup Language ("lenguaje de etiquetas extensible") es un metalenguaje desarrollado por el World Wide Web Consortium (W3C) [W3C, 2002]. XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Ventajas del XML:

*Es extensible, lo que quiere decir que una vez diseñado un lenguaje y puesto en producción, igual es posible extenderlo con la adición de nuevas etiquetas de manera de que los antiguos consumidores de la vieja versión todavía puedan entender el nuevo formato.

*El analizador es un componente estándar, no es necesario crear un analizador específico para cada lenguaje. Esto posibilita el empleo de uno de los tantos disponibles. De esta manera se evitan bugs y se acelera el desarrollo de la aplicación.

*Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarlo. Mejora la compatibilidad entre aplicaciones.

Referencias

[Abramson, 1991] Abramson, D. Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms. 1991.

[Brelaz,1979] . Brélaz, D. New methods to color vertices of a graph. 1979.

[Chartrand,1993] Gary Chartrand & Ortrud R. Oellermann, Applied and Algorithmic Graph Theory, McGraw-Hill, Inc. 1993

[Colorni,1992] Colorni A., Dorigo M., Maniezzo V. A genetic algorithm to solve the timetable problem. Italia. 1992.

[Coello-Coello,2006]. Introducción a la computación evolutiva. 2006.

[Costa, 1994] Costa, D. A Tabu Search Algorithm for Computing an Operational Timetable. 1994.

[Culberson] J. Culberson. Una Implementación del algoritmo DSATUR. http://www.cs.ualberta.ca/~joe/Coloring/index.html

[De Werra, 1985] De Werra D. “An introduction to timetabling”. 1985.

[DIM93] DIMACS. Satisfiability Suggested Format. 1993. www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/satformat.ps

[DIMACS] DIMACS. Clique and Coloring Problems Graph Format. 1993.

[DPLL,1962]. M. Davis, G. Logemann, D. Loveland. A machine program for theorem proving. 1962.

[Even, 1976] Even S, Itai A, Shamir A. “On the complexity of timetabling and multicommodity”. 1976.

[Garey, 1979] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., San Francisco, CA, 1979.

[Glover,1993]. Glover F. & Laguna, M. Tabu Search. Modern Heuristic Techniques for Combinatorial Problems. 1993.

[Gross,1998] Jonathan Gross & Jay Yellen, Graph Theory and Its

Applications.

1998.

[Hamilton,1981]. A.G. Hamilton . Lógica para Matemáticos. 1981.

[Johnson-Aragon,1991] Johnson D, Aragon C. Optimization by simulated annealing: part II, graph coloring and number partitioning. 1991.

[Junginger,1986] Junginger W. Timetabling in Germany A Survey. 1986

[Kenneth-Chin Fat, 2004]. Kenneth K., Chin-A-Fat. School timetabling using Satisfiability Solvers. Netherlands. 2004.

[Kleer, 1992] An improved incremental algorithm for generating prime implicates. 1992.

[Kirkpatrick, 1983] Kirkpatrick S. Gelatt D. “Optimization by Simulted Annealing”. 1983.

[Lalescu,2003]. Liviu Lalescu. Timetabling experiments using genetic algorithms. Rumania. 2003.

[Lee,2000]. H.S.C.Lee, Timetabling Highly Constrained Systems via Genetic Algorithms, Masters Thesis, University of Philippines. 2000.

[Metaheuristics] www.metaheuristics.net

[Molina, 2007]. Molina Araya, J. Algoritmos Evolutivos para la resolución de un problema de tipo Timetabling. 2007.

[Neufel-Tartar,1974] Neufeld G., Tartar J. Graph coloring conditions for the existence of solutions to the timetable problem. 1974.

[PATAT] http://www.asap.cs.nott.ac.uk/ASAP/ttg/patat-index.html

[Redl,2004]. Timothy Anton Redl. A Study of Univerity Timetabling that Blends Graph Coloring. Rice University. 2004.

[Ross, 1997]. Ross P. , Corne D. Some observations about GA-based exam timetabling. PATAT Second Conference. 1997.

[SAT2004]. Competencia SAT 2004. 2004. http://www.satlive.org/index.jsp

[Schaerf,1995]. Andrea Schaerf. A survey of Automated Timetabling. CWI,

Netherlands.

[Swee-Chuan,2003] Swee-Chuan Tan. An object oriented timetabling framework with applications. Monash University. 2003.

[Triska, 2008] Solution Methods for the Social Golfer Problem

Universidad Tecnológica de Viena. 2008.

Simsttab. Markus Triska. 2005.
http://stud4.tuwien.ac.at/~e0225855/index.html

[W3C,2002] W3C 2002. World-Wide Web Consortium XML. http://www.w3.org/

[Weiss, 1997] Weiss M. “Data structures and algorithm analysis in C”. 1997.

[Zhang]. Hantao Zhang. Finite Model Generation. http://www.cs.uiowa.edu/~hzhang/sato/sato4.1tgz

[Zhang, 1999] Zhang, H. Stickel, M. Implementing the Davis-Putnam Method. 1999.